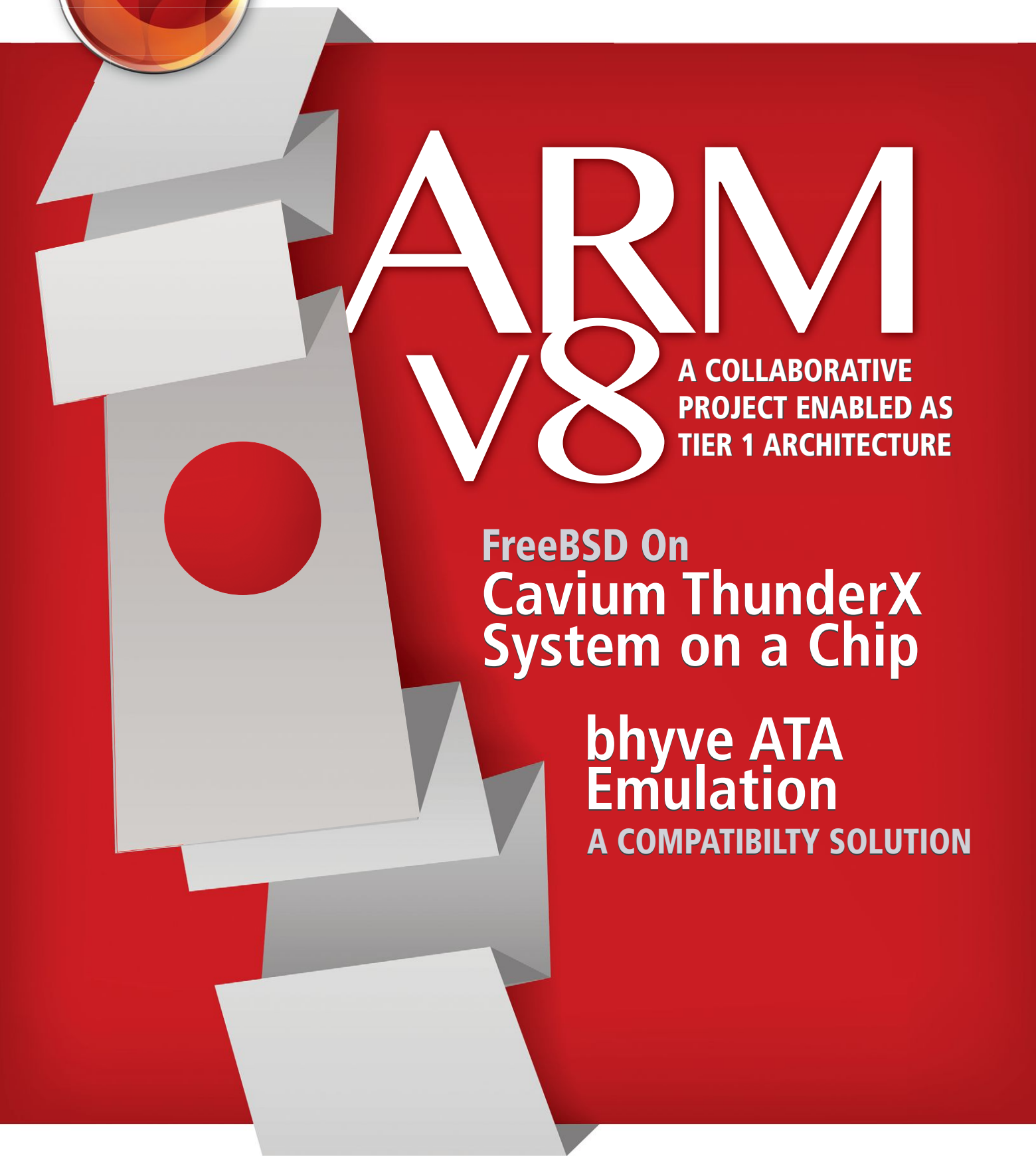
FreeBSD™ JOURNAL
May/June 2016



ARM v8

A COLLABORATIVE PROJECT ENABLED AS TIER 1 ARCHITECTURE

FreeBSD On Cavium ThunderX System on a Chip

bhyve ATA Emulation

A COMPATIBILTY SOLUTION

Introducing the new
XG-2758 1U pfSense® Security Appliance

Fast 10 Gigabit networking at a price you can afford.

XG-2758 1U features include:

- 8 Core Intel® Atom™ C2758 2.4 GHz with AES-NI and Quick Assist Technology.
- 16GB ECC RAM
- 4x 1Gb Ethernet RJ45 via Intel i354 on-chip; 1 port configurable RJ45 or SFP.
- 2x 10Gb Ethernet SFP+ via Intel 82599 Niantic.
- Optional PCIe x8 slot available for further expansion.
- Preloaded with pfSense Open Source software. No maintenance, licensing or upgrade fees.
- Flexible Configuration - firewall, LAN or WAN router, VPN appliance, DHCP Server, DNS Server, multi-WAN and high availability.
- Fully extendable with add-on software packages such as Snort®, Squid, SquidGuard, Suricata, to enable IDS/IPS, load balancing, traffic optimization, reporting and monitoring.
- Create VPNs to the Amazon Cloud easily with our with Amazon® AWS™ Wizard.

pfSense®

Shop now at the official pfSense store or authorized partners worldwide.
http://store.pfsense.org/XG-2758

FreeBSD™ JOURNAL May/June 2016

Table of Contents

SOMETHING XL
IS COMING IN APRIL

ENTERPRISE-CLASS HARDWARE, RUNNING THE WORLD'S MOST POPULAR OPEN SOURCE STORAGE OPERATING SYSTEM.

For more information on the FreeNAS Mini, visit **iXsystems.com/mini** today.

Then, check back in April for something XL...

S&W PUBLISHING LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Art Director • Dianne M. Kischitz
dianne@freebsdjournal.com

Office Administrator • Michael Davis
davism@freebsdjournal.com

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0)
is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).

Published by the FreeBSD Foundation,
PO Box 20247, Boulder, CO 80308
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org

LETTER from the Board

FreeBSD is even better ARMed for the latest 64-bit architecture.

In this issue we have two articles covering recent work on the ARMv8 architecture, which is ARM Inc.'s foray into the world of 64-bit server computing. FreeBSD was an early adopter of the ARMv8 architecture with the FreeBSD Foundation, ARM Inc., and Cavium all pitching in money and engineering resources to make sure FreeBSD was properly supported on this new architecture. Andrew Wafaa describes what it took to get FreeBSD enabled as a Tier 1 architecture on ARMv8. A Tier 1 architecture is one that is self-hosting and has a complete set of 3rd-party packages from the FreeBSD Ports collection. Making it to Tier 1 is an important milestone for any architecture on which FreeBSD runs, and indicates a serious level of commitment that will carry over for many years.

Zbigniew Bodek and Wojciech Macek discuss FreeBSD running on a particular implementation of the ARMv8 architecture, Cavium's ThunderX platform. ARM does not make chips itself; it licenses the designs for others to produce chips, and to add and remove interesting features from those chips. This level of specialization is what has made ARM so popular in mobile and embedded platforms. A hardware manufacturer only takes the parts beyond the basic core that they need to build their platform. In order to bring up FreeBSD on real hardware, rather than a processor simulation, there has to be some hardware, and because Cavium stepped forward with their platform early in the process, it was the first hardware to be targeted by FreeBSD developers. Zbigniew Bodek and Wojciech Macek do not work for Cavium directly, but worked with them to bring up FreeBSD on the ThunderX platform.

Our third article covers an important new feature in FreeBSD's native virtualization solution—bhyve. Teaca Ionut-Alexandru, Mihai Carabas, and Peter Grehan describe their work with providing bhyve with an ATA emulation layer. ATA and ATAPI emulation are necessary to support older FreeBSD guests on top of modern FreeBSD hosts. Many people run FreeBSD applications on older releases for years, and having a virtual platform available for these legacy applications is an important transition path for many companies.

In his svn update column, Steven Kreuzer talks about all the changes that have gone into the FreeBSD tree due to the work on ARM. He has taken up the torch on this column and is clearly off and running with it.

Dru Lavigne interviews Benedict Reuschling, who has been contributing to FreeBSD in many ways over the years, and is now a member, along with Dru, of the FreeBSD Foundation's Board of Directors. Benedict teaches in Darmstadt, Germany, and has a passion for moving FreeBSD into more classrooms.

Lastly, it's always great to hear from our readers, in email or in person. If you see one of the editorial board members at an upcoming conference—and there are several more to attend in 2016—make sure to let us know what you think about the *Journal* and how it can better serve the FreeBSD and general computing communities.

Sincerely, George Neville-Neil
For the *FreeBSD Journal* Editorial Board

SEE TEXT ONLY

ARMv8

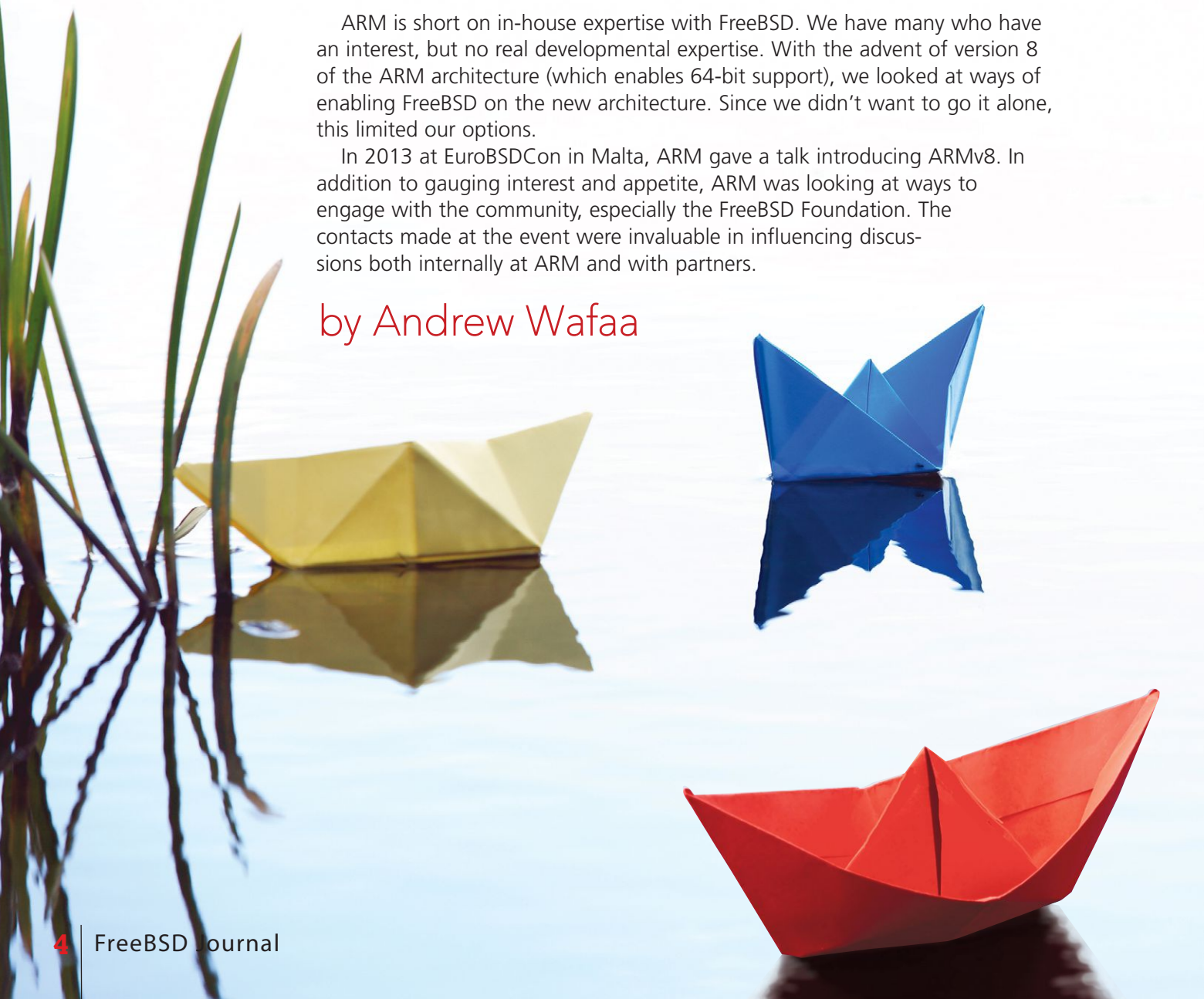
ENABLED AS TIER 1 ARCHITECTURE: A COLLABORATIVE DEVELOPMENT PROJECT

FreeBSD support for the ARM architecture has historically been a grassroots affair, and for a whole host of embedded platforms this works well. With the launch of ARMv8, the development of ARM-based systems has expanded into the enterprise space, both from a server infrastructure and a network infrastructure standpoint. To support the enterprise market, it was clear we needed a more aggressive development model to deliver the most complete and competitive offering of FreeBSD.

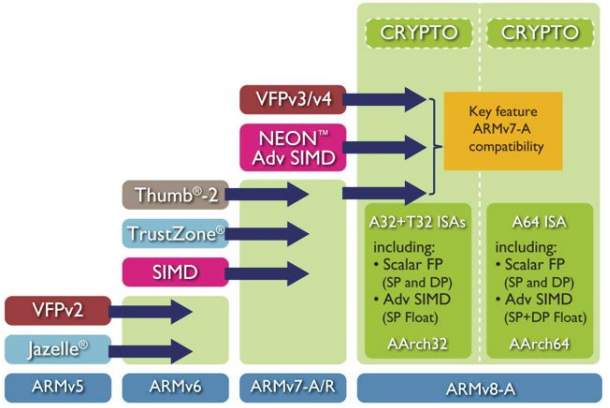
ARM is short on in-house expertise with FreeBSD. We have many who have an interest, but no real developmental expertise. With the advent of version 8 of the ARM architecture (which enables 64-bit support), we looked at ways of enabling FreeBSD on the new architecture. Since we didn't want to go it alone, this limited our options.

In 2013 at EuroBSDCon in Malta, ARM gave a talk introducing ARMv8. In addition to gauging interest and appetite, ARM was looking at ways to engage with the community, especially the FreeBSD Foundation. The contacts made at the event were invaluable in influencing discussions both internally at ARM and with partners.

by Andrew Wafaa



Time ticked on, and in 2014 ARM was confident enough to take the plunge and truly engage with the community in getting ARMv8 supported on FreeBSD as a Tier 1 architecture. ARM decided the best path forward was to sponsor the FreeBSD Foundation and work with them to get the right people, including Cavium Inc., engaged to move forward with enablement. As these discussions were going on, Andrew Turner, a longtime FreeBSD developer and committer, began work on getting 64-bit ARM (known as AArch64 or ARMv8 or ARM64) support into FreeBSD. ARM has been in discussions with Semihalf since 2012 on the topic of 64-bit ARM, Cavium started discussions in 2013 with Semihalf, and it was clear to both ARM and Cavium that Semihalf was a partner that needed to be involved.



Overview of the ARM Architecture

On October 1, 2014, Cavium released a public announcement describing the partnership with ARM and the FreeBSD Foundation. Both ARM and Cavium provided funding to the Foundation, which in turn engaged Andrew Turner and Semihalf. By working with and through the Foundation, both ARM and Cavium wanted to ensure that the effort was an inclusive and collaborative affair.

A TRUE COLLABORATION

As some people may not know all the parties involved, here is a brief overview of the key entities:

ARM Ltd. is an IP design house with a comprehensive product offering, including 32-bit and 64-bit RISC microprocessors, graphics processors, enabling software, cell libraries, embedded memories, high-speed connectivity products, peripherals, and development tools.

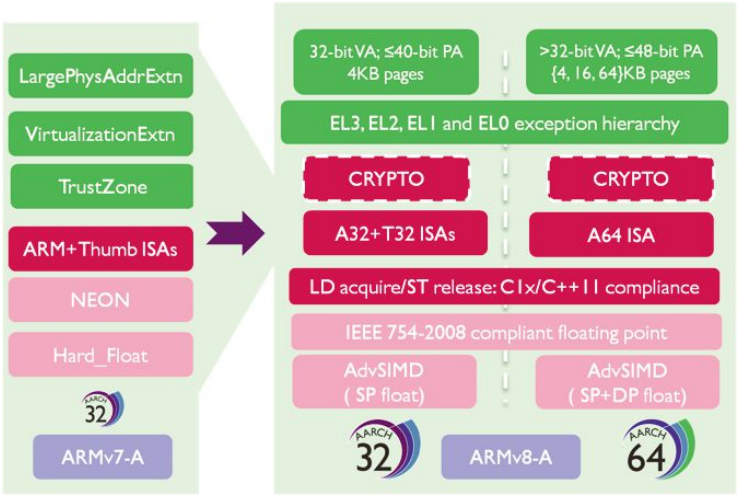
Cavium (NASDAQ: CAVM) is a leading provider of highly integrated semiconductor products that enable intelligent processing in enterprise, data center, cloud, and wired and wireless service provider applications. Cavium offers a broad portfolio of integrated, software-compatible processors ranging in performance from 100 Mbps to 100 Gbps that enable secure, intelligent functionality in enterprise, data-center, broadband/consumer and access and service provider equipment. In 2014 Cavium launched the ThunderX® Workload Optimized SOC focused on server workloads and scale out Data Center deployments.

Semihalf creates software for advanced solutions in the areas of operating systems, virtualization, networking, and storage. They make software tightly coupled with the underlying hardware to achieve maximum system capacity. Their partners and customers are semiconductor industry leaders. The following team at Semihalf was responsible for ThunderX/ARM64 development work: Wojciech Macek (project leader, FreeBSD committer; responsible for ARMv8 base support, PCI, SMP and storage areas), Zbigniew Bodek (FreeBSD committer; ARMv8 base support, GICv3/ITS, VNIC), Dominik Ermel (AHCI, testing), Michał Stanek (PCI, SMP, debugging), and Tomasz Nowicki (KDB, watchpoints), with additional help from Michał Mazur (UEFI bring-up and debug).

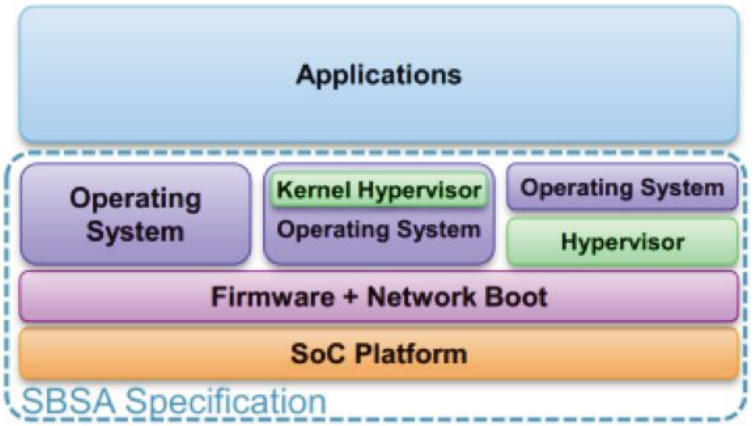
Andrew Turner is a freelance software engineer and FreeBSD committer focusing on running FreeBSD on ARM-based chips. He started the ARM64 FreeBSD port. Andrew was responsible for the overall system architecture, the parts of the system that may be used by many SoCs. Prior to this, Andrew worked as an embedded software engineer for consulting companies and silicon vendors.

Divide and Conquer

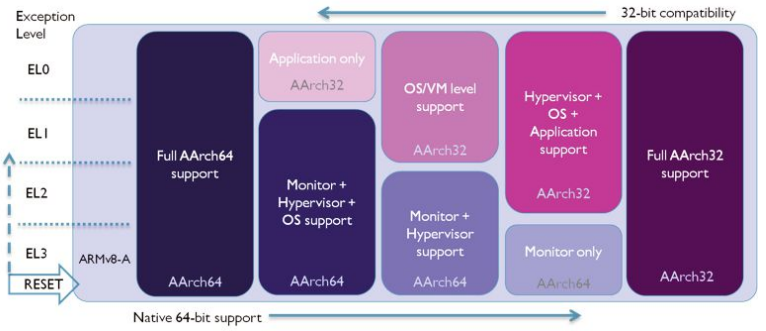
As ARMv8 is a new architecture and not an extension of previous ones, it was agreed to break the required work into three tangible pieces that comprised Kernel, Userland, and Platform. ARM was on hand to provide support to Andrew and Semihalf, and naturally enabled both parties with the required tools—develop-



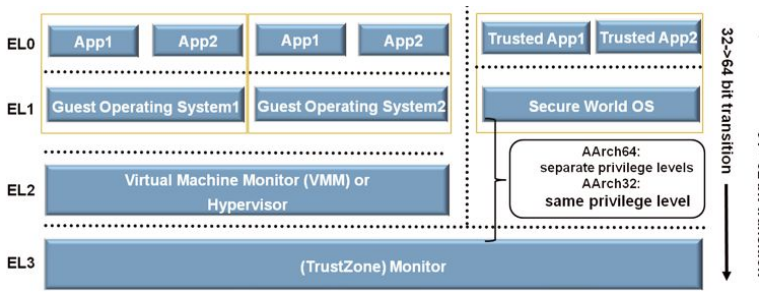
Evolution of the ARM Architecture



Overview of SBSA



ARMv8-A exceptional levels



Hyervisor exceptional levels

ment platforms, debugging, and documentation. As a silicon vendor, Cavium was able to provide Semihalf with access to early development boards and initial development servers to the Sentex colocation facility, enabling more developers access to server hardware.

Vanquishing Runaway Differentiation

This effort was seen as foundational and the basic enabler for the ARM and FreeBSD ecosystems to build upon. As such, ARM wanted to make sure that it was as standards compliant as possible, to help ensure there was no fragmentation (something that ARM was accused of in the past). This meant ensuring support and compliance with the ARM Server Base System Architecture (SBSA), the ARM Server Base Boot Requirements (SBBR), Power State Control Interface (PSCI), and the VM Specification.

The SBSA is a hardware specification for Firmware and OS developers. It enables application developers to target a single OS image for all ARMv8-A systems. OS and Firmware vendors benefit from having a well-defined platform to target, and a single OS image will work across all compliant systems regardless of vendor or micro-architecture.

The SBBR is a follow-on companion to the SBSA. It defines the platform firmware abstractions necessary for OS deployment and boot, hardware configuration and management, and power control of SBSA-compliant systems. It covers items such as UEFI, ACPI, and Trusted Firmware.

PSCI defines a standard interface for power management that can be used by OS vendors for supervisory software working at different levels of privilege. Operating systems, hypervisors, and secure firmware must interoperate when power is being managed; this standard should ease the integration between supervisory software from different vendors working at different privilege levels.

The VM specification is targeted at images for virtualized guests, ensuring that images are portable across hypervisors.

In addition to being compliant with the above specifications, other items included:

- Basic CPU support
  - Initial target of ARMv8, including the Cortex®-A57 and ThunderX microarchitectures
- Environment
  - Toolchain
  - Loader
- Peripherals
  - Timers, GIC (Global Interrupt Controller)
  - Block storage
- ARM locore
  - Assembly macros & definitions
  - Bootstrap
  - Exception handling
  - Cache maintenance and context switching
- VM / pmap
  - TLB maintenance and Page fault routines
  - Conversion of pmap to 64-bit
- 64bit userspace glue
  - Syscalls, signals handling, libraries entrypoint
- SMP
- Dual Socket w/ Cache Coherency

Starting the FreeBSD ARMv8 Port

Work started in earnest just after summer of 2014, and by spring of 2015 it was possible to log into FreeBSD on an AArch64 system. Support was still very basic, and there were some pieces missing. A key component missing was DTrace. Under advice from the Foundation, ARM engaged the University of Cambridge to work on implementing DTrace and Hardware Performance Counter support. This work greatly increased the options for debugging and helped build a more complete FreeBSD story. At BSDCan 2015, Semihalf demonstrated the progress that had been made by showing FreeBSD running on a 48-core Cavium ThunderX platform.

FreeBSD on ARM, prior to the advent of the ARM64 port, relied on u-boot as a key part of the boot flow. While u-boot has the dominant position in this role for the embedded space, UEFI-compliant boot solutions find a more prominent role in the enterprise/networking space. As such, UEFI compliance was chosen as the default boot loader requirement. The FreeBSD loader was ported to be an EFI

application that could be loaded and executed by UEFI-compliant boot loaders.

One of the more challenging areas was in the area of CPU scalability. A lot of blood, sweat, and tears were shed by Semihalf and Andrew Turner in getting Cavium's lowest core count (48) supported, especially when it came to static assumptions in the kernel, which were resolved case by case. Semihalf's Zbigniew Bodek performed a live demo on Cavium's 48-core ThunderX system at the June 2015 FreeBSD developer summit held at the BSDCan conference in Ottawa, Canada.

Together we have made tremendous progress. In the interest of continuing improvement, all parties are still working together to solidify and improve the earlier work. ARM and Cavium continue to work with our partners and the FreeBSD community and sponsor the FreeBSD Foundation.

George Neville-Neil, a member of the FreeBSD Foundation board, said, "The work done to bring ARMv8 to FreeBSD has been a significant undertaking that could not have happened without the support of both ARM and Cavium, who, working with the FreeBSD Foundation, were able to get a brand new architecture up and running in record time. ARMv8 is now a significant architecture within the FreeBSD ecosystem and continues to progress along with the rest of the operating system and its tools."

The Path to FreeBSD 11

We are currently preparing the FreeBSD/arm64 port to be included in the upcoming FreeBSD 11.0 release, addressing the remaining attributes required of a Tier-1 architecture. This includes updates to the documentation, running the FreeBSD test suite and addressing failures, sourcing and installing hardware for the FreeBSD release engineering team and security officer, and validating the install procedures.

The FreeBSD release engineering team produces AArch64 snapshot install media and virtual machine images. Snapshots are announced on the *freebsd-snapshots* mailing list, and are available from ftp://ftp.freebsd.org/pub/FreeBSD/snapshots/VM-IMAGES/11.0-CURRENT/aarch64/Latest/.

For those who would like to test FreeBSD on AArch64, there are a few options. One could use emulation with QEMU. The following steps detail how to go about this:

- Install the QEMU emulator pkg, or build from ports

```
% pkg install qemu-devel
```

- Fetch the snapshot virtual machine image

```
% fetch ftp:// ftp.freebsd.org/pub/FreeBSD/snapshots/VM-IMAGES/11.0-CURRENT/aarch64/Latest/
FreeBSD-11.0-CURRENT-arm64-aarch64.qcow2.xz
```

- Fetch a special version of the UEFI boot firmware

```
% fetch http://people.FreeBSD.org/~gjb/QEMU_EFI.fd
```

- Uncompress the vm image

```
% xz -d FreeBSD-11.0-CURRENT-arm64-aarch64.qcow2.xz
```

- Start the virtual machine

```
% qemu-system-aarch64 -m 4096M -cpu cortex-a57 -M virt -bios QEMU_EFI.fd -serial telnet::
4444,server -nographic -drive if=none,file=FreeBSD-11.0-CURRENT-arm64-aarch64.qcow2,id=hd0 -device
virtio-blk-device,drive=hd0 -device virtio-net-device,netdev=net0 -netdev user,id=net0
```

- Connect to the virtual machine console with telnet

```
% telnet localhost 4444
```

Emulation is fine for some use cases, but nothing beats real hardware. A variety of hardware platforms are becoming available at various price points and capabilities. Inexpensive (under $250 USD) single board systems are available from the 96Boards project, including the HiSilicon-powered HiKey board and the Qualcomm Dragonboard. Cavium, AMD, and Applied Micro provide higher-spec processors and systems.

Given its strong feature set and availability, Cavium's ThunderX platform is the primary hardware reference platform for FreeBSD's ARMv8 port. Instructions for bringing up FreeBSD/arm64 on the ThunderX can be found at https://wiki.freebsd.org/arm64/ThunderX.

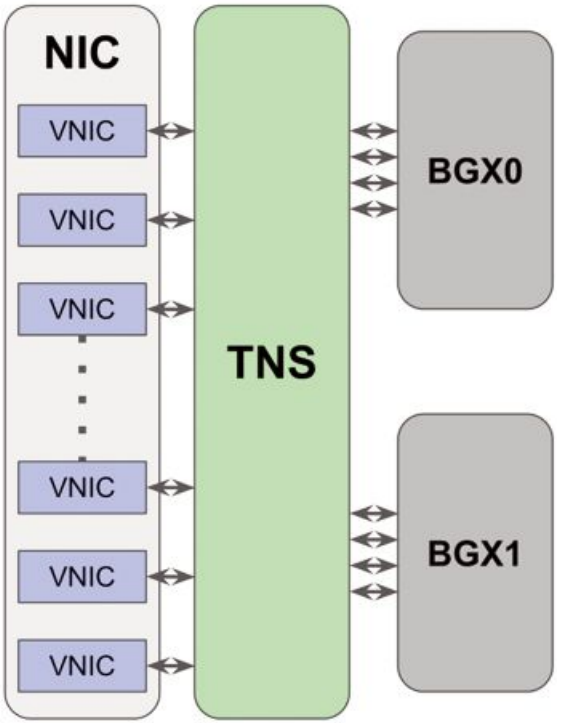
Cavium ThunderX, the Primary Reference Platform

Cavium's ThunderX platform is available in two configurations— a single socket 48-core platform and a dual socket 96-core variant.

ThunderX supports a wide variety of features and peripheral devices and also offers powerful Ethernet capabilities that include 40 Gbps, 20/10 Gbps, and 1 Gbps interfaces. The networking subsystem is partitioned into a few core components:

- IBGX - Common Ethernet Interface
- INIC - Networking Interface Controller
- ITNS - Traffic Network Switch

In the presented order, these implement: MAC layer, Network Interface layer, and hardware switching between the mentioned entities and



High level view of the network subsystem in ThunderX

other on-chip devices. Moreover, NIC is a SR-IOV (Single Root I/O Virtualization) capable device that can incorporate up to 128 Virtual Functions, each providing full network interface features.

In order to support ThunderX networking, FreeBSD introduces drivers for NIC, VNIC, BGX, and MDIO interface (used for the slower connection types such as SGMII for 1Gb Ethernet). The contributed work is available in mainline FreeBSD under sys/dev/vnic/ directory. The core part of the FreeBSD network interface support is located in nicvf_main.c and nicvf_queues.c files that handle

Virtual Function (VF) operations. VFs are able to perform DMA transactions to and from the main memory in order to transfer packet traffic. Each VNIC contains a set of hardware queues (8 send, receive, completion, and 2 Rx buffer) that can be freely used by other VFs if the current one is not enabled. This gives some interesting parallelization capabilities that can be utilized by the OS.

> "A lot of blood, sweat, and tears were shed BY SEMIHALF AND ANDREW TURNER IN GETTING CAVIUM'S LOWEST CORE COUNT (48) SUPPORTED"

The driver was tested with 1, 10, and 40 Gbps connections, and currently the system is capable of saturating 10 Gbps link on Tx path, using 4 CPU cores. The exemplary benchmark that can provide such results is iperf3(1), testing TCP connections using 4 parallel threads. Linerate can be achieved in both copy and zero-copy configurations between userspace and kernel.

This was possible thanks to enabling hardware capabilities such as L3, L4 checksum calculation offloading, TCP Segmentation Offload (TSO), and OS features including Large Receive Offload (LRO). Other Semihalf optimizations seriously improved networking and general I/O throughput. The next step is to further upgrade NIC multicore scalability by enabling additional hardware queues beyond standard sets of 8.

In addition to the ThunderX-based systems, Cavium's OCTEON TX CN80/81XX IOT Gateway/Router and 4-Bay NAS reference designs will be available soon and priced under U.S. $800. These systems are based on the 64-bit ARM-based dual and quad core SoCs that are optimized for low power networking, security, IoT, and control plane applications.

Future Work

There are a number of additional tasks in planning, in order to improve FreeBSD and make the best use of AArch64's capabilities.

1. Improvements to the resource constraints framework, in order to better support OS-level virtualization.

2. Exploration of new scheduling algorithms, taking advantage of FreeBSD's pluggable scheduler. This makes it viable to easily do a completely _new_ scheduler aiming at mobile, for example. The interfaces are also very clean and bereft of all the messy bits that Linux has accumulated over the years.

3. Improved NUMA support in the VM and the scheduler, to improve scalability.

4. Beginning of the implementation of a power management subsystem with more generalized CPU frequency scaling support.

5. CPU count scalability, by profiling bottlenecks using DTrace and hwpmc. This is more from an analysis perspective using some of the "pedigree" tooling that FreeBSD has (such as DTrace for example—SystemTap on Linux is still considered experimental) to figure out how FreeBSD is scaling and where the bottlenecks exist. There is a conversation with David Chisnall and ARM on a similar themed track for Google SoC. Getting analysis is a great first step to then chalk out plans to fix/add features.

6. Power management is still lacking, and ARM has ideas on how best this could be implemented and is discussing these with members of the community. •

Reference Links

Cavium ThunderX Homepage
FreeBSD Foundation Projects
ARM Cortex-A series Homepage
ARMv8 Architecture Reference Manual
Semihalf Homepage

With over 16 years' experience in Open Source, Andrew Wafaa's role at ARM is to ensure that ARM architecture is as well supported within open-source projects as possible. This takes his interactions through the full stack from operating systems to userland applications, and all points in between. A key component is fostering relationships and facilitating getting the job done.

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.
freebsdfoundation.org/donate

FreeBSD™
FOUNDATION

SEE TEXT ONLY

FreeBSD on CAVIUM THUNDERX SYSTEM ON A CHIP

By Zbigniew Bodek and Wojciech Macek

THIS ARTICLE DESCRIBES THE FREEBSD OPERATING SYSTEM PORT FOR THE CAVIUM THUNDERX CN88XX SYSTEM ON A CHIP. THUNDERX IS A NEWLY INTRODUCED ARM64 (ARMV8) SOC DESIGNED FOR THE HIGH-PERFORMANCE AND SERVER MARKETS. IT IS CURRENTLY THE ONLY ONE IN THE ARM WORLD TO INCORPORATE UP TO 96 CPU CORES IN THE SYSTEM ALONG WITH THE TECHNOLOGY TO MAKE IT POSSIBLE.

ThunderX is up to date with the latest trends in the computer architecture industry, including those that are relatively new to FreeBSD, like SR-IOV (Single Root I/O Virtualization), or completely unique, such as ARM GICv3 and ITS.

The main focus here is to provide a bottom-up view of how the FreeBSD platform support for ThunderX was implemented and to depict the benefits and pitfalls of the newly introduced ARMv8 technology in terms of the OS development. The article also describes key components of the ThunderX system and explains how they were supported in FreeBSD. Finally, possible areas of further improvements are pointed out briefly.

INTRODUCTION

FreeBSD is undoubtedly the most recognizable Unix-like operating system available. One of the main areas of its deployment is the server market, which is still dominated by Intel and AMD-based computers. However, recently in the mobility industry, the highly successful ARM architecture is gaining interest as a foundation for high-performance server SoCs. A turning point in that field was the emergence of a 64-bit ARM implementation (ARMv8) including improved technology to obtain insane multi-core capabilities. Thus, it is important for the FreeBSD community (developers and users) to keep up with the growing interest in ARM-based servers and supply the ecosystem with an ARM64 BSD OS that will be on par with the available Tier-1 x86 platforms.

The motivation behind the work on ThunderX was driven by the actual market need for the FreeBSD OS on that platform, the aim of which is

to become a real alternative for energy-intensive solutions. ThunderX is also the only ARM-based chip in the world to scale up to 48 CPU cores per socket with a possible dual socket configuration (up to 96 CPUs). With the included hardware accelerators for networking, storage, and security, as well as powerful peripheral devices, ThunderX is a perfect base for the server-oriented OS such as FreeBSD and a great engineering challenge for a kernel developer.

The support introduced here is based on foundational work with emulation as a primary target (ARM Foundation Model). Hence the platform support for ThunderX was partly carried out in parallel to the FreeBSD base system development and occasionally was intertwined with it. ThunderX was also the first hardware platform to switch from the ARM emulator. The result is a fully functional OS that supports key chip features such as:

- PCI Express
- GICv3 + ITS
- SMP (single and dual socket)
- SATA
- Virtualized Network Interfaces

The description of the general FreeBSD kernel implementation for ARMv8 architecture is beyond the scope of this paper and is discussed only with respect to the work on ThunderX support.

HARDWARE OVERVIEW

Contemporary ARM-based chips very much follow current trends in the computer industry, incorporating multiprocessor capabilities; high-performance, peripheral devices; buses and extensions, as well as various hardware accelerators and virtualization technologies. The 8th architecture revision (ARMv8) moved those concepts to a new level by overcoming previous architectural limitations. ThunderX is a good representative of the new ARM chips generation, as it is the first to utilize majority of the newly introduced features.

Core Complex and CCPI

The heart of the CN88XX SoC is a set of Cavium proprietary ARMv8-compliant CPUs. Each CPU has its own I-cache and D-cache but they share the common 16MB L2 cache. A single package can contain up to 48 CPUs organized in three clusters of 16 CPUs each. The core complex can be connected to another CN88XX processor using Cavium Coherent Processor Interconnect fabric (CCPI). All CPUs in the system are cache coherent in respect to L1, L2 cache, and DMA accesses. The system coherency capabilities are also extended across CCPI-connected entities. Therefore, the dual socket configuration yields a fully coherent system containing 96 ThunderX CPUs. The potential multi-socket configuration can incorporate four nodes with a total number of 192 CPUs [1].

PORTING

The efforts of porting the FreeBSD operating system to a new platform can usually be divided into a few general stages:

- Toolchain and build environment support.
- Kernel loading and bootstrapping.
- `locore.S` and low-level operations. Kernel start code and low-level machine bring-up as well as basic cache maintenance, atomic operations, synchronization primitives, etc.
- Elementary system operations. These include virtual memory management in `pmap.c`, exceptions handling, context switching, and interrupts as well as system timers and console support.
- Drivers for peripheral devices.
- Userland support.

Most of the listed steps had already been implemented in FreeBSD, as a lot of work has been put into running it on Qemu and ARM Foundation Model. However, the real hardware was still a huge unexplored area. Existing pieces of the code provided a good starting point for ThunderX support. The missing parts of the development mainly concerned:

- System bootstrap.
  Every real hardware device has its own quirks, assumptions, and, most of all, custom firmware. Even though ThunderX is an ARMv8-compliant platform, it's in fact a custom ARM implementation, which requires some more care than generic Cortex- A53/ A57 tested on Qemu and ARM emulators.
- Interrupts handling.
  A massive number of CPU cores and peripheral devices, as well as the PCI-centric architecture demanded a completely new approach to interrupts signaling and a number of changes to both generic ARM64- and ThunderX-specif ic code.

CONTINUES NEXT PAGE

- PCI-express.
  The connectivity between the ThunderX core complex and peripheral coprocessors is almost entirely based on the PCIe bus. Support for the on-chip PCIe bridges hierarchy was critical in relation to other integrated interfaces such as SATA, network or USB.
- A complex network controller.
  A powerful networking card with IO virtualization and modular architecture.
- SMP operation with 48/96 CPU cores. Running FreeBSD on a real hardware and in a multi-core environment revealed a list of issues that needed to be investigated and resolved.

SYSTEM BOOTSTRAP

A typical CN88XX boot scenario starts with the on-chip Boot ROM. This stage is supposed to load Boot-level 1 Firmware (BL1) from the SPI-connected FLASH, which will then load further BL2 and BL3 Firmware, and, finally, the control over the system is passed to the Cavium Unified Extensible Firmware Interface (UEFI) bootloader. At that point the ThunderX system and its interfaces are initialized and the boot CPU core is in EL2 exception level (Hypervisor). In order to run FreeBSD, ThunderX needs to load the kernel ELF file and supply it with a machine description such as the DTB (Device Tree Blob), available memory regions, as well as information about the kernel location in DRAM, etc. Hence the next step in the FreeBSD boot process is the native BSD *loader(8)* which in that case is executed in a UEFI runtime environment. *loader(8)* handles kernel acquisition and jumps into its code. ThunderX uses genuine ARM64 *loader(8)*, so almost no modifications needed to be made.

EARLY SYSTEM INITIALIZATION

The very first kernel code being executed is the one in `locore.S`. It performs three fundamental actions:

- Puts CPU into a well-defined state.
- Prepares an execution environment for the C code.
- Forwards information about the modules. obtained from the bootloader.

The ARMv8 processors (in AArch64 state) operate in one of the four maximum Exception Levels: EL0-EL3, from which EL0 has the lowest software execution privilege that corresponds to the *User Mode*; EL1 can be called a *Kernel Mode*, EL2 is a *Hypervisor*, and EL3 is used for ARM's TrustZone *Secure Monitor Mode* [3]. The start code usually drops the exception level to EL1 (after performing an EL2-specific configuration). At this point, the initial and identity (1:1) kernel mappings are created, which will allow changing the context to the kernel virtual address space when the Memory Management Unit is enabled. Before that happens, all settings related to address translation, caching, and initial system behavior (such as exceptions reception) are applied. Finally, the kernel stack is configured and CPU jumps to the early machine initialization in C code.

ThunderX requires more settings during this stage than the ARM Foundation Model. These include:

- Enabling EL1 access to the Generic Interrupt Controller's CPU Interface.
  By default, the CPU interface cannot be accessed through System Registers from EL1. Access permission has to be granted while still in EL2.
- Enlargement of the virtual address space in `TCR_EL1`.
  On some variants of ThunderX, the physical memory is mapped beyond 512GB. Therefore, if the programmed address space is not big enough, it is impossible to create an identity mapping required to jump from the physical to the virtual address space.

The changes, however, are not strictly ThunderX-specific and will apply to any platform with similar requirements.

INTERRUPTS DELIVERY

Exceptions whose purpose is to indicate to CPU that a certain action took place are called interrupts. There would be no reasonable multi-threading OS without the interrupts support; therefore, they are one of the most fundamental elements of the system. Previous generations of ARM processors often incorporated a so-called ARM Generic Interrupt Controller (GIC) or other proprietary implementation.

Exemplary ARM GIC consists of two main components: Distributor and CPU Interfaces. Typically, interrupt lines from the on-chip devices are wired to the distributor interface, which then, according to its configuration, routes the interrupt signals to the appropriate CPU interfaces. If the interrupt signaling is enabled, the CPU will receive a notification on the appropriate interrupt line (IRQ or FIQ). Finally, the CPU can acquire the interrupt information, such as its number, from the CPU Interface registers and change the pending state

of the interrupt. This architecture works fine for ARMv6/v7 processors but has some serious limitations in terms of:

- Scalability.
  Can route interrupts up to 8 CPUcores.
- Maximum number of interrupts.
  Each interrupt requires a physical connection to the distributor.
- No Message Signaled Interrupts support.
  Cannot use in-band PCI interrupt signaling.
- Slow access to the CPU Interface registers.
  Each interrupt requires at least a few read/write sequences to the memory-mapped registers (slow access to device memory and possible TLB misses).

Generic Interrupt Controller v3

Platforms such as ThunderX require improved interrupts handling to provide better SMP utilization, support for PCIe devices, and minimal time penalty per interrupt. These features were introduced with ARM Generic Interrupt Controller v3 [2] in cooperation with the Interrupt Translation Service. The contributed work includes full FreeBSD support for ARM GICv3 and ITS along with the ThunderX-specific quirks, but excluding virtualization extensions.

This article describes the support for the crucial GIC components only and does not cover the implementation of the machine-dependent part of the interrupts handling code that needed to be redesigned for the purpose of this port.

Affinity-based Routing

Unlike earlier GIC architectures, GICv3 incorporates an additional, third component in the form of Re-Distributors, which are memory-mapped entities associated with every CPU in the system. Moreover, the CPU Interface can now be accessed through the CPU's System Registers to speed up interrupts handling after the core gets notification. To overcome the interrupt-to-CPU delivery limitations, the interrupt is now routed based on the Affinity Hierarchy. This means that the interrupt destination is now addressed by the 4-level CPU affinity number in the system. The GICv3 driver configures all SPIs (Shared Peripheral Interrupts) in the global Distributor, but PPIs (Private Peripheral Interrupts), SGIs (Software Generated Interrupts), and a new class of LPIs (Local Peripheral Interrupts) are managed through per-CPU Re-Distributors. Each Re-Distributor needs to be enabled or woken up before it can be used. Fortunately, GICv3 provides an auto-configuration scheme that allows the OS driver to iterate through a device's memory-mapped region, match the configurator CPU to a correct Re-Distributor (based on the affinity), and perform appropriate actions. Once configured, the Re-Distributor interface can be used in a similar manner to the global Distributor.

Inter Processor Interrupts (IPI) can also be delivered based on the CPU Affinity Hierarchy. Because the FreeBSD kernel does not enumerate CPUs in SMP according to their hardware affinity, it is required to save and match each CPU address with the requested CPU group on every IPI. Software Generated Interrupts (triggered by the write to the CPU Interface register) are used to perform IPI exchange.

ITS and Message-Signaled Interrupts

In a typical scenario the peripheral device requests an interrupt in the Distributor that then forwards it to the appropriate Re-Distributor. Finally, the interrupt is signaled to the CPU Interface. The alternative behavior, for which Re-Distributors are used in GICv3, is interrupt routing that bypasses the Distributor. This is used by MSIs and requires Interrupt Translation Service (ITS) assist. Both ways are depicted in Figure 1.

ITS is a GICv3 extension that manages routing and migration of LPIs generated by any device that can send Message-Signaled Interrupts. The unique approach presented by the ITS controller is that all MSI-capable devices can use a single memory location (`GITS_TRANSLATER` register address) to generate an interrupt. The interrupt request number and appropriate routing is deduced based on the interrupting device's identifier and translation information programmed into the ITS. The interrupt controller works closely with the PCIe bus and IOMMU because unique device IDs used in the interrupt translation process are passed within the bus transaction itself. The programming of the ITS is performed via commands sent to the special Command Queue and is required to set up the relation between the PCI endpoint, LPI numbers (a.k.a.

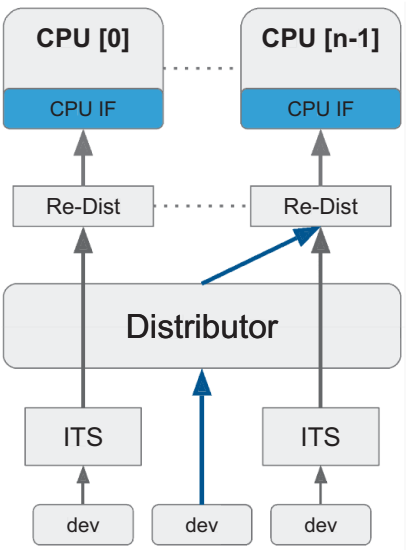


Fig. 1: Interrupts distribution using GICv3 and ITS.

interrupts collection), and the target Re-Distributor.

The FreeBSD support for ITS consists of the GICv3 subordinate driver (`gic_v3_its.c`) and device methods for allocating and mapping MSIs (`pic_alloc_{msi, msix}(), pic_map_msi()`). The controller requires some portion of the system memory to operate and this needs to be provided by the OS. The variety of possible ITS implementations implies the existence of auto-detection features that need to be revised when configuring the device. This includes not only the amount of RAM to reserve for the controller, but also various cacheability/shareability attributes, as well as page sizes used by the ITS memory system.

The absolute basic configuration requires:

- **Memory for ITS Private Tables** Private controller's tables that are not accessible to the programmer.
- **Memory for the Command Queue** A ring buffer for the ITS commands. Apart from the memory reservation, OS software must set the queue write pointer (`GITS_CWRITER`) to the start of the queue.
- **Memory for the LPI Configuration and Pending Tables** The LPIs pending status is visible through the bitmap in the Pending Tables. A particular LPI can be configured (i.e., masked/unmasked) using an array of bytes in the Configuration Table.

Interrupt mapping is created per device and per Re-Distributor (so in fact per CPU). The implementation-defined device identifiers are usually based on the PCIe *Bus:Device:Function* address. For ThunderX, this ID is more complicated due to possible multi-socket configuration and requires the additional *Node:ECAM.number* address. Device IDs differ for the internal and external PCIe units and this needs to be taken into consideration by the ITS code. CPUs on the other hand are matched based on their CPU affnity or Re-Distributor physical address.

In order to create an LPI interrupt route the software has to:

- **Map interrupts collection to a Re-Distributor (`MAPC` command)** Managing interrupts via collections and not stand-alone entities is useful when migrating LPIs from one CPU to another. This, however, does not occur very often on FreeBSD, but still is required as a part of interrupts routing. An ITS driver assigns collection IDs based on the destination CPU IDs, so the subsequent interrupt-to-collection mapping can be easily associated with the interrupt-to-CPU assignment.
- **Allocate and assign an Interrupt Translation Table (ITT) to a device (`MAPD` command)** The software has to provision memory for an array of Translation Table Entries; each mapping interrupts for a given device. ITT will be associated with the device ID after issuing `MAPD` command to the ITS.
- **Reserve a range of LPI numbers** Each MSI vector is signaled to the CPU as an LPI. The pool of available LPI *physical* numbers starts with 8192 and is configurable through an array of bytes in memory. Usually MSI-capable endpoints will request a range of vectors rather than a single interrupt. For that reason, the driver needs to find a free range of LPIs and allocate it for a particular device. LPI bookkeeping is achieved by a bitmap in which each bit represents an interrupt number. This maps directly to the pool of free LPIs and portions of the bitmap are assigned for each MSI-capable device.
- **Map MSI vector to the LPI, device ID, and a collection (`MAPVI` command)** The benefit of the interrupt translation is that any virtual interrupt number that is being sent by the device can be mapped to a given *physical* interrupt vector (LPI). Therefore, each endpoint can simply use any convenient combination of messages. The `MAPVI` command inserts an interrupt translation and route to the appropriate CPU into the ITT of the selected device. This final step provides a full set of information that allows for MSI delivery.

The ITS driver's PIC methods provide the implementation of the described steps. `PIC_ALLOC_MSI/MSIX` allocates the abstract *ITS device*, which is in fact an Interrupt Translation Table with a set of pre-allocated LPI numbers. The main difference between MSI and MSIX allocation is that MSIs are requested as a range of vectors, whereas MSIX vectors are requested one at a time. Common `PIC_MAP_MSI` callback does exactly that: it maps an MSI vector to the LPI for a given device. Finally, the LPI needs to be unmasked in a Configuration Table; however, this has to be initiated from the top level PIC, which is, in that case, the GICv3 driver.

Although very extensive, this design provides a

CONTINUES NEXT COLUMN

flexible way of managing Message-Signaled Interrupts and simplifies the interrupt resources' assignment for each device by providing a single MSI/MSIX triggering address, arbitrary selection of MSI data, hardware translation to a *physical* interrupt identifier, and a large number of supported devices and interrupt vectors.

SYMMETRIC MULTIPROCESSING

The standard ARMv7-MP specification limits the number of supported CPU cores to eight. Each four cores are logically connected to create a cluster. Then, up to two clusters can be combined using the CoreLink interface that provides a fully coherent 8-CPU core system. Some vendors' implementations of ARMv7 cores can provide up to 16-cluster scalability, which seemed to be the theoretical limit for the architecture. ARMv8 is a huge step forward. The interconnects now are able to address each CPU by its logical location using four-level CPU *Affnity* Address (*A 3:A 2:A 1:A 0*) that allows a significant growth in total core number.

SMP Bring-up

ThunderX CPU cores are managed through a standard ARM Power State Coordination Interface (PSCI). The relevant code was already available in FreeBSD sources and was used as is. The main work done to support SMP operation on ThunderX was focused on resolving problems in areas such as:

- System and TLB cache management
- IPI and interrupts handling
- Context switching
- Memory ordering
- Operations atomicity

For example, the system maintains cache coherence between CPU cores, but only within their shareability domain. If the common memory mappings are not marked as shareable in that domain, data copies seen by the CPUs may differ. Similar results can be observed when one CPU modifies shared Translation Table entries and appropriate TLB maintenance operation is not issued and propagated to the secondary cores. In that case, CPUs can potentially see different physical frames with different access permissions at the same virtual addresses.

CCPI and Dual Socket Operation

ThunderX chips provide even more sophisticated scalability. Based on the Cavium Coherent Processor Interconnect (CCPI), two processors can be connected creating a shared memory space. The typical two-socket configuration supports 96 cores and up to 1 TB of system memory. From the operating-system perspective, the complete machine looks like it has two separate NUMA (Non-Uniform Memory Access) nodes, each made of 48 CPUs and half of the memory. The I/O interfaces (e.g., PCIe, SATA) are accessible by both nodes, but it is strongly suggested that all I/O accesses be done by the socket owning the corresponding interface. In that scenario, the entire system performance and peripherals are doubled. The dual-socket machine offers twice as many PCIe links, Ethernet interfaces, etc., and the interrupts are distributed among all CPUs using two separate ITS units. For even bigger workloads, the two-socket Cavium systems can be connected using a low-latency Ethernet fabric. This allows for hundreds of gigabits per second of aggregated network bandwidth.

PCIE

The Cavium ThunderX machine provides a standardized interface to which all peripherals are attached. The only I/O the CPU provides is a modern PCIe 3.0 bus. All other devices (SATA, Ethernet NICs, etc.) are typical PCIe endpoints that can be easily detected by the operating system and do not require any machine-specific resource management code except for a single PCIe controller driver. ThunderX provides two distinct PCIe interface types: internal and external. The internal one is a reduced version of the generic PCIe standard providing PCI-like logical access to all peripherals inside the ThunderX SoC. The external one, on the other hand, is a fully compatible PCIe 3.0 link allowing easy connection of generic PCIe cards of up to x8 lanes.

The ThunderX driver is divided into three parts: generic PCIe hardware accessors, FDT-configured internal PCIe controller, and, finally, an internal PCIe device representing an external PCIe controller. The FreeBSD PCI subsystem takes care of almost every aspect of PCI operation except for some very hardware-dependent, low-level functionalities. In order to fulfill these requirements, the driver needs to provide three things: access to a device's configuration space, resource (bus addresses) allocations, and interrupt mapping. On the CN88XX platform, any access to the internal configuration space is done using a generic mechanism called ECAM (i.e., all configuration headers of devices are mapped into the host memory space; each memory access to that location causes the controller to automatically generate all PCIe requests for the user). External

PCIe configuration space is accessed using indirect addressing supported by an external controller. Second, the functionality that the driver needs to provide is a resource assignment. Fortunately, the Cavium UEFI configures all PCIe trees and fills in every BAR with appropriate values. The only thing the driver needs to do is read those (bus) addresses, mark them as used in the Resource Manager (*rman(9)*), and return a result. If a driver is not initialized (this happens for example for NIC's Virtual Functions), the controller manually allocates the necessary bus space and properly configures the BAR. The last thing the driver provides is interrupt mapping. Currently, the only supported interrupt types are MSI or MSI-X, which require some quirks in ITS as well.

The advanced architecture of ThunderX offers a huge amount of internal PCIe devices (more than 200 endpoints). To avoid creation of enormous PCIe device trees, these devices are separated into three different zones, each of which is governed by a separate internal PCIe controller. It is also worth noting that the external PCIe controller is the PCIe device attached to the internal PCIe bus. Although not intuitive, this solution provides an easy way to support various hardware versions of the device. Let's imagine one ThunderX chip has only one external PCIe available, whereas another might have three. Using a conventional approach, each version of the hardware would require a different machine description (i.e., DTB) file to make all the controllers get detected and configured properly. But when the external controller is an internal PCIe device, a number of them are gathered on-the-fly using a standard PCIe enumeration technique.

VNIC

The CN88XX chip has powerful Ethernet capabilities that include 40 Gbps, 20/10 Gbps, and 1 Gbps interfaces. ThunderX introduces a flexible and highly programmable design of the network subsystem that allows for efficient hardware resource virtualization and node-to-node connectivity without using external switches. The main objective of the presented work was to provide a basic networking support for all types of available interfaces.

The networking subsystem in ThunderX is partitioned into a few, core components

- BGX - Common Ethernet Interface
- NIC - Network Interface Controller
- TNS - Traffic Network Switch

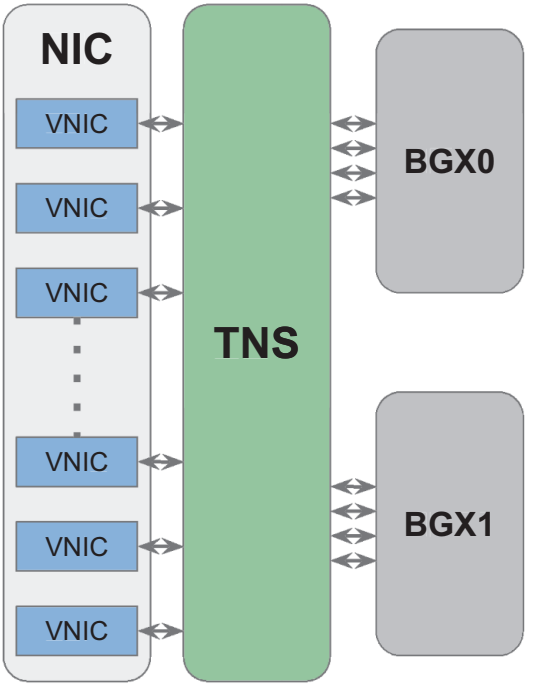


Fig. 2: Network subsystem architecture.

that, in the presented order, implement: the MAC layer, the Network Interface layer, and hardware switching between the mentioned components and other CN88XX devices. The high-level view of network subsystem architecture is depicted in Figure 2. Moreover, NIC is a SR-IOV [4] capable device, that can incorporate up to 128 Virtual Functions, each providing full network interface features. ThunderX contains 2 BGX instances that are connected to the NIC and its VFs via a TNS unit. However, the described FreeBSD implementation of the ThunderX networking drivers does not include support for the TNS and its features. TNS Bypass logic is used instead, which results in the direct connection of BGX units to the corresponding NIC TNS interfaces as well as freedom of Rx/Tx queues assignment to Logical MACs provided by the BGX.

The contributed work is located in the `sys/dev/vnic/` directory in the sources tree and consists of the group of drivers, each implementing the BGX, NIC, VNIC, and MDIO interface. The MDIO setup and partially the BGX configuration are based on the *FDT* description, currently the only method available.

BGX

The Programmable MAC layer is implemented in the BGX controller. It is seen as a regular PCIe endpoint and can be configured without the explicit machine description provided; however, BGX-to-Ethernet PHYs assignment needs to be

presented to the driver. Each of the two built-in BGX controllers can provide up to 4 Logical MACs (LMACs) with maximum rate of 10 Gbps each, or a single 40 Gbps LMAC. Any LMAC can be connected to the arbitrary NIC's Virtual Function. LMAC types are selected by the low-level firmware and FreeBSD driver retrieves this information to proceed with the appropriate setup.

The core BGX code, located in the `thunder_bgx.c` file, is responsible for the general interface bring-up, such as SerDes configuration, detection of the LMAC type, and final Ethernet interface configuration. Since BGX can be attached to a variety of Ethernet PHYs, different media connection types need to be supported. High-speed interfaces, such as XLAUI, XAUI, DXAUI, or XFI, are managed from the BGX driver; however, low-speed SGMII uses a dedicated MDIO driver located in the `thunder_mdio.c` file. The latter exports *KOBJ(9)* methods for PHY connection management (`LMAC_PHY_{CONNECT, DISCONNECT}`) as well as link state polling (`LMAC_MEDIA_STATUS`).

During a normal BGX driver operation, software polls the MAC layer status to keep the NIC Physical Function up to date. The polling itself is done from the *callout(9)* context and is executed periodically (once per two system ticks). BGX/LMAC re-configuration is performed upon link status change, such as speed transition, etc. Current link status is stored in the driver's software context for each LMAC and is exported to the PF driver on demand.

Physical Function

The Physical Function driver, located in `nic_main.c`, cooperates with BGXes and TNS to create a highly programmable network interface. Unlike other popular NIC cards, the CN88XX Physical Function does not provide networking capabilities, but rather is a resource manager for subordinate Virtual Functions (VFs) and an interface between the MAC layer (BGX) and the networking interface layer (VNIC). PF supports up to 128 Virtual Functions using PCI SR-IOV [4] technology. The communication between PF and VFs is held using private *Mailboxes* for each VF. Any changes in the MAC layer or configuration requests are signaled to the VFs using *Mailbox* interrupt. The Physical Function receives requests from the VFs in a similar manner.

The introduced FreeBSD driver uses a generic PCI IOV subsystem to create and configure Virtual Functions. The key step of the VF bring-up is the `pci_iov_attach()` invocation. This is done in PF's `nic_sriov_init()` function, called during the device attach procedure. At this point, PF's and VF's so-called configuration schemes (high-level interface options, such as MAC address selection capabilities) are specified and passed to the PCI IOV subsystem. The code also needs to supply the PCI layer with the *IOV KOBJ(9)* methods, such as:

- `PCI_IOV_INIT(9)`
  Implemented by `nicpf_iov_init()`, validates the number of requested VFs and saves this value for later use when the actual bring-up occurs.
- `PCI_IOV_UNINIT(9)`
  Is supposed to disable previously enabled VFs.
- `PCI_IOV_ADD_VF(9)`
  Is called when SR-IOV infrastructure is initializing a new VF. Options requested by the `pci_iov_attach()`, can be used to set up the VF's parameters based on the configuration schema.

During a normal driver's operation, the PF's code periodically polls LMACs' links through the BGX interface and handles VF↔PF requests, which may relate to the VF's Queues configuration, link/MAC status changing, etc.

Virtual Function

Virtual Functions implement the networking capabilities of VNIC. VFs are able to perform DMA transactions to and from the main memory in order to transfer packet traffic. The presented VNIC driver consists of the two logically separated parts: `nicvf_main.c` and `nicvf_queues.c`. The first one performs the typical FreeBSD's network interface configuration and provides *ifnet(9)* callbacks, such as `if_init`, `if_ioctl`, etc. As the controller can handle more than one transmitting queue, the driver implements a multi-queue variant of the Tx path, which uses the `nicvf_if_transmit()` function for the outgoing traffic. The Ethernet media status is updated through the messages from the PF, so a stub `nicvf_media_status()` function just exports this information to the upper layers. The driver also supports hardware and interface statistics that are refreshed periodically in the `nicvf_tick_stats()` callout.

The second part of the driver, located in the `nicvf_queues.c`, is responsible for the controller's resource allocation as well as packets transmission and reception. In particular `nicvf_config_data_transfer()` is the

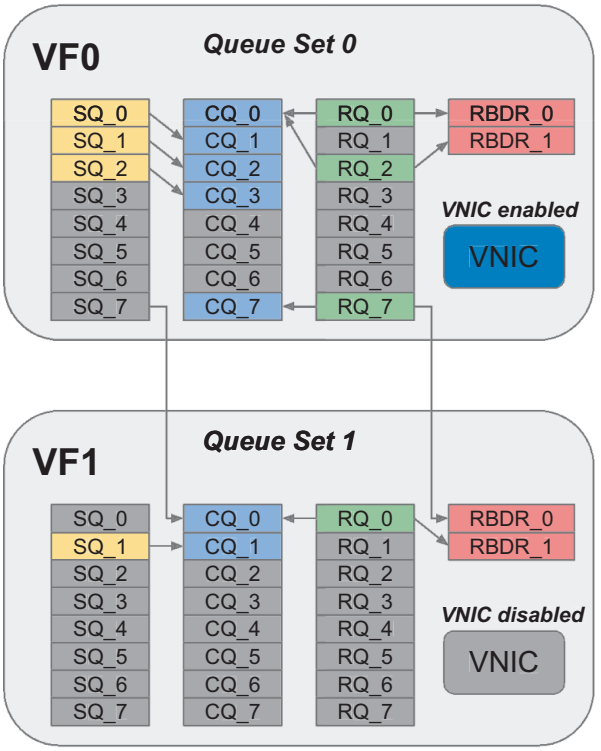


Fig. 3: Exemplary NIC's QS configuration

focal point of the VF's queues configuration.

Each Virtual Function contains a Queue Set (QS) that consists of:

- 8 x Completion Queue (CQ)
- 8 x Send Queue (SQ)
- 8 x Receive Queue (RQ)
- 2 x Receive Buffer Ring (RBDR)

The Completion Queue contains data describing all completed actions, such as packet transmission or reception. Other queues are assigned by the Physical Function to the selected CQs, potentially many-to-one.

The transmitter side uses SQ to describe the egress data and actions that need to be performed on the packet before it can be sent (e.g., L3, L4 checksums calculation). Each SQ is subscribed to a target Completion Queue.

Receive Queues are VNIC's internal structures that describe how to receive packets. They need CQ and RBDR assignments. More than one RQ can be attached to both CQ and RBDR. RBDRs on the other hand, describe free buffers in the main memory that can be used to store received data. Upon packet reception, VNIC moves the incoming data to the free buffer provided by the RBDR descriptor and returns its physical addresses to the assigned Completion Queue. This could possibly be a drawback, as it is much more difficult to locate a received buffer in the memory using physical addressing rather than virtual. However, this was resolved by allocating memory exclusively from the *Direct Map*, which is a continuous region of linearly mapped memory. Thanks to that, it is fairly simple to find the received buffer in the virtual address space. Moreover, parts of the ingress buffers, used by the RBDR queues, are adapted to store a small descriptor containing *mbuf(9)* information as well as *bus_dma(9)* related data. This approach can significantly simplify *mbufs* bookkeeping and reduce *mbuf* access latency on Rx.

An interesting capability of the NIC's Queue Sets is that if the VNIC on a particular VF is not enabled, its queues can still be used by another operational VNIC. This makes it possible to assign one Queue Set to a single VNIC, all Queue Sets to a single VNIC, or anything in between. An exemplary, yet possible QS configuration is depicted in Figure 3. Currently, the VF driver utilizes single Queue Set per VNIC by default.

AVAILABILITY

Support has been integrated into the mainline FreeBSD 11.0-CURRENT and is publicly available starting with SVN revision no. 289550. As of the writing of this article, work on improving FreeBSD on ThunderX is still in progress.

Future Development

Even though FreeBSD support for ThunderX is in decent shape, there are a number of areas that could benefit from further development. There are also other functional issues that need to be resolved before dual-socket configuration can be fully supported in mainline FreeBSD. Some more interesting areas of development are briefly described below:

- **Dual-socket Support in Mainline FreeBSD**

To improve kernel performance, all physical addresses from `DMAP_MIN_PHYSADDR` to `DMAP_MAX_PHYSADDR` are mapped statically into a continuous VA region called DMAP (Direct Map). When the PA space is fragmented (as it is for ThunderX in dual-socket configuration), the VA DMAP range is huge and exceeds the limit that the CPU core is able to address in 3-level Translation Table mode. Semihalf implemented a patch that resolves those issues by creating multiple regions of DMAP range to save space and allow for dual-socket device operation. However, the lookups in DMAP arrays are likely to reduce the performance, and hence the 4-level Translation Table solution is the preferred one. However, the current 4-level Page Tables implementation does not allow for successful dual-socket boot. This is

CONTINUES NEXT PAGE

an area that needs to be reworked in order to support two ThunderX nodes in the mainline.

Multiple ITS Support

Currently, only the ITS attached to socket-0 is operational, but in the case of a dual socket system, this resource is doubled. FreeBSD/arm64 lacks support for multiple and chained interrupt controllers; therefore, changes in the interrupts handling code for ARM64 architecture need to be applied.

ACKNOWLEDGMENTS

Special thanks to the following people:

- Dominik Ermel, Michał Mazur, Tomasz Nowicki, and Michał Stanek (Semihalf) for their work and commitment to this project.
- Ed Maste (The FreeBSD Foundation), Andrew Wafaa (ARM), and Larry Wikelius (Cavium) for their successful cooperation.
- Andrew Turner (FreeBSD Project) for insightful reviews and advice.
- Rafał Jaworowski (Semihalf) for organizing and managing this project.
- The success of this project is a joint work of the Semihalf team, Andrew Turner, ARM, Cavium, and The FreeBSD Foundation. The project was sponsored by ARM, Cavium, The FreeBSD Foundation, and Semihalf.●

Zbigniew Bodek, a Software Engineer at Semihalf, focuses on BSD and Linux operating systems development for ARM and PowerPC-based computers. His main areas of interest are computer science, microprocessor technology, embedded systems and kernel development. He has been a FreeBSD committer since 2013. For the last 1.5 years Zbigniew has been involved in FreeBSD development and optimizations for multi-core ARMv8 chips.

Wojciech Macek is a Software Engineer at Semihalf, a small company in a frosty part of the world. He is primarily interested in ARM architecture, kernel internals and ultra-fast storage solutions. His recent work has focused on enhancing and optimizing FreeBSD for multi-core ARMv8 systems.

REFERENCES

[1] Cavium, *Cavium ThunderX CN88XX Hardware Reference Manual*. (2015)

[2] ARM Ltd. Processor Division, *GIC Architecture Specification PRD03-GENC- 010745 12.0.* (2013)

[3] ARM Ltd., *ARM Architecture Reference Manual for ARMv8-A architecture profile*. (2013–2015)

[4] Intel, *PCI-SIG SR-IOV Primer, 321211-002*. (2011)

SEE TEXT ONLY

bhyve ATA Emulation

The bhyve ATA/ATAPI emulation is part of a larger project that aims to ensure backward compatibility with older versions of FreeBSD guests for Free BSD Hypervisor (bhyve). Currently the bhyve hypervisor emulates AHCI standard for drive and atapi devices. In order to support guests like FreeBSD 4 that do not have ahci drivers, it is necessary to emulate an ATA/ATAPI controller.

by Teaca Ionut-Alexandru,
Mihai Carabas, and
Peter Grehan

Introduction

In this article we present the emulation of a generic ATA drive controller connected on the PCI bus through a Host PCI Adapter. Both ATA controller and Host Adapter are parts of our implementation that is designed and developed from scratch. Using this emulator, we simulate an ATA disk controller that is used by the guest ATA driver from the bhyve virtual machine.

Currently bhyve supports any version of FreeBSD i386/amd64 since the FreeBSD 8.0 release. The standard AHCI mode has also been supported out of the box on FreeBSD since version 8.0. The scope of this article presents a solution to provide compatibility of guest operating systems with older versions such as FreeBSD4/5. Consequently, emulation of the ATA Host Adapter Standard in bhyve hypervisor is required to support guest operating systems that have drivers only for the ATA controllers.

We begin by presenting some of the related work in ATA emulation, the motivation for starting to write this driver from scratch. We continue with an overview of device emulation in bhyve hypervisor. Most of the devices are emulated in userspace in usr.sbin/bhyve. The other ones, such as the Programmable Interrupt Controllers (PIC) and the timers, are implemented into the kernel in vmm/io. There are two categories of devices emulated in bhyve, the ISA and PCI devices. Through the ISA devices we enumerate the UART controller and RTC (real-time clock) controller. One part of the PCI devices is represented by the virtio class containing block, net, and rng (random entropy from /dev/random) subclasses. Also the AHCI controller is a PCI device emulated in bhyve. Our implementation of ATA Host Adapter will be emulated as a PCI device through a register to the PCI controller. That is also emulated in the bhyve hypervisor.

We present some general information about bhyve and some technical standards such as PATA, SATA, AHCI, and PCI that are related to the ATA emulation.

The ATA (AT Attachment) defines the physical, electrical, transport, and command protocols for the internal attachment of storage devices to host systems [4]. There can be Parallel ATA (PATA) or Serial ATA (SATA) interface standards. Parallel ATA (PATA) is the legacy AT Attachment, being an interface standard for the connection of storage devices like hard disks, floppy drives, and optical disc drives in computers [7]. Serial ATA takes the place of the former legacy AT Attachment standard, offering many advantages over the older interface: reduced cable size and cost (seven conductors instead of 40 or 80), native hot swapping, faster data transfer through higher signaling rates, and more efficient transfer through an (optional) I/O queuing protocol [8].

The Advanced Host Controller Interface (AHCI) is a host adapter for the Serial ATA disk drive controller. This specification defines the functional behavior and software interface of the Advanced Host Controller Interface, which is a hardware device that is an interface communication between the software and Serial ATA devices. AHCI is a PCI class device that performs movement data between the system host memory and Serial ATA devices [1].

For the Parallel ATA protocol there is also a host adapter controller interface. The ATA/ATAPI Host Adapters Standard specifies the AT Attachment Interface between host systems and storage devices using Direct Memory Access protocol. The AT Attachment Interface can be used in any host system that has a PCI bus and storage devices connected to the processor [2].

Overview of the bhyve Hypervisor Structure

bhyve stands for BSD hypervisor and is a hypervisor/virtual machine manager introduced into the FreeBSD operating system. It is similar to Linux KVM in that it runs on the host OS and relies on modern CPU features such as Intel VT-x, Extended Page Tables, and VirtIO network and storage drivers.

bhyve is comprised of the vmm.ko loadable kernel module, the libvmmapi library, and the bhyve, bhyveload, and bhyvectl utilities. To use any of these utilities, the vmm.ko module must be loaded first. The bhyveload tool helps load a FreeBSD kernel from a disk image. For example, */usr/sbin/bhyveload -m 256 -d ./vm0.img vm0.*

It shows the FreeBSD loader screen and you should see the device /dev/vmm/vm0.

The bhyve tool is used to boot the VM with 2 vCPUs, the same 256M RAM and the tap0 network interface.

/usr/sbin/bhyve -c 2 -m 256 -A -H -P -s 0:0,hostbridge-s 1:0,virtio-net,tap0 -s 2:0, ahci-hd,./vm0.img -s 31,lpc -l com1,stdio vm0

After the VM has been shut down, its resources can be reclaimed with:

/usr/sbin/bhyvectl –destroy –vm=vm0

The bhyve process starts by initializing the pci controller in the init_pci function. This function begins the bus enumeration to find all PCI devices. It iterates through each bus, each slot, and each function to find if there is a PCI device. For example, if the input parameter of the bhyve program is "3:0,ahci- hd,./diskdev" (that means bnum = 0, snum = 3, fnum = 0), the PCI controller associates this combination to a custom device, and maps it with a pci_devemu structure that has a pointer to the callback which initializes its PCI device, in this case, the 'pci_ahci_hd_init' function. By calling this function, the ahci device is initialized by providing identification information (the vendor and device information and the class, subclass, and progif information) to the pci_devinst structure. The PCI emulator maps the (bus, slot, function) combination with the pci_devinst structure, and, in so doing, the identification information from the pci_devinst structure belongs to the device controller. Also, the pci_ahci_hd_init callback is allocating the BAR register with baridx = 5 and type = PCIBAR MEM32. The ahci driver from the guest operating system programs the Base Address Registers to inform the ahci device of its address mapping by writing configuration commands to the PCI controller.

Related Work

There are no other related implementations with the ATA Host Adapter Standard emulated in the bhyve hypervisor. There is an ATA controller emulator implemented in the GXemul framework that supports full-system computer architecture emulation. It is, however, almost impossible to port this software in the bhyve sources tree due to the fact that it uses a different application interface for communication with the rest of the system and because it is coded in the C++ language. But we could use this software for a better understanding of the ATA protocols implemented there, like PIO, DMA, or other information regarding the ATA commands.

There is an implementation of the Advanced Host Controller Interface standard emulated in bhyve. In order to understand the mechanisms used in the emulation of the AHC standard, documentation of this implementation needs to be done.

In order to catch the host commands that are sent to the AHCI controller, the pci_ahci module in bhyve registers two handlers, pci_ahci_write and pci_ahci_read, that are called whenever the host software tries to address the ahci device through the BAR registers. Basically these callbacks read/write the value from the address computed by the baridx = 5 register and an offset. The implementations of these callbacks are dependent on the interface controller (ahci/ata/atapi). They emulate by accessing the iovec array from the prdt and writing to the diskdev file descriptor. To complete, the command will call the ioctl system call for emulating an interrupt. The io requests are created through these callbacks. The blockif_req requests are processed by the blockif thread in the ata_ioreq_cb callback. The ata_ioreq_cb routine is completed through an ioctl system call to the vmm.

For the manipulation of io requests there are two routines, blockif_dequeue and blockif_enqueue. First there is an io request array of elements and free/inuse queues grouped in a blockif_ctxt structure. The blockif_dequeue routine is called from the blockif thread context and it tails the first element from the inuse queue and updates the free/inuse queues. The blockif_enqueue gets a free element, fills it with a blockif_req request, and updates the free/inuse queues accordingly.

The DMA Write/Read requests are handled by the ahci_handle_dma implementation that builds up the iovec based on the Physical Region Descriptor Table (prdt). These iovec requests are processed in blockif_proc that briefly writes the iovec array to the diskdev file descriptor using the pwritev system call. The DMA command finishes by triggering an interrupt.

Architecture

In this section we focus on the most important design concepts at the base of implementation. First, we explain the primary/secondary and master/slave relationships followed by the register descriptions involved in the emulator interface and finishing with the interrupt-based mechanism that is used in driver-emulator communication and is the reason why the ATA emulator approaches an event-driven architecture.

Master Drives and Slave Drives

Before presenting the architecture of the ATA controller, let's take a look at the primary/secondary and master/slave relationships and the way the ATA emulator will be configured.

Most motherboards have two IDE interfaces (primary and secondary), also known as channels. In both primary and secondary IDE channels, only ATA can be connected to, which means that IDE only supports ATA/ATAPI drives. Each interface could support two devices, for a total of up to four drives. The two drives have to decide for themselves how to share the same ATA channel. To accomplish this, one drive on each channel is designated as the "master" and the other drive (if present) is designated the "slave." So that leaves us with the following possibilities:

- Primary Master Drive
- Primary Slave Drive
- Secondary Master Drive
- Secondary Slave Drive

Each drive can be either ATA or ATAPI.

Our PCI ATA adapter is emulating for the ATA legacy driver located in `sys/dev/ata/ata-pci.c` under the FreeBSD code base. After some investigation of the ATA legacy driver, we have observed that it is supporting only one channel, meaning only two drives. For more details, please see the 'int_ata legacy(device_t dev)' function from the `sys/dev/ata/ata-pci`.c driver device.

With these in mind, we choose to configure the ATA emulator using these parameters:

-s N:M,ata-hd,X,./DISK_MASTER,./DISK_SLAVE

or

-s N:M,ata-hd,X,./DISK_MASTER

where N:M is the pci slot information, X is 0 or 1 (Primary or Secondary channel) followed by the name of the disks, the first one being the master drive, and the second the slave drive. There might be only one disk representing the master drive. In any case, the emulator implementation is supporting two channels, and holding different data structures for both of the channels, even though the guest legacy driver is using only one (Primary or Secondary).

PCI and I/O Register Descriptions

In order to emulate the ATA controller disk, two sets of registers should be implemented as the controller is going to be emulated through the PCI bus, and so the commands sent by the guest driver are provided using this interface, the PCI interface. The set of adapter registers represent the PCI space configuration which the guest driver reads after the enumeration phase to detect controller configuration details. See the Implementation section of this article for details on the configuration of these registers.

The ATA controller also contains a set of registers (ATA registers). Basically this set is configured by the guest driver to communicate with the drive controller. The implementation of the emulator should provide such a register interface in addition to the data structure necessary to maintain the controller states and to emulate the driver commands. The ATA controller registers are divided into two categories. The first represents the ATA Bus Master Registers. The bus master ATA function uses 16 bytes of I/O space. All bus master ATA I/O space registers can be accessed as byte, word, or double word quantities. The base address for these registers is PCI BAR 4 [3]. The second one represents ATA Channel Registers.

The Command Block registers are used for sending commands to the device or posting status from the device. These registers include the LBA High, LBA Mid, Device, Sector Count, Command, Status, Features, Error, and Data registers. The Control Block registers are used for device control and to post alternate status. These registers include the Device Control and Alternate Status registers [5]. For the primary channel, these registers are addressed by the BAR0 and BAR1 registers, and the secondary channel uses BAR2 and BAR3.

Interrupts

The design of the ATA emulator is based on an event-driven architecture, where the events represent the write operations on the emulator registers that are actually the guest driver commands. After these commands are interpreted and processed, the state of the emulator is updated accordingly. When the command is fully emulated, the guest driver is notified by asserting an interrupt. The emulator adapter is supporting two channels conforming to the primary and secondary channel address and have a separate IRQ for each channel. A parallel ATA controller is using the 14 and 15 IRQs. The emulator is reserving these IRQ numbers in the initialization phase and asserts an IRQ at each command completion.

Block Device Emulation

For better management of the disk operations like read/write calls, the ATA emulator uses the implementation for block device emulations in bhyve. An instance of the blockif_ctxt structure is associated with each disk drive. Basically the ATA emulator supports a maximum of two disk drives—the master and slave drives. Hence, each of them will be assigned one blockif_ctxt structure. In addition to the design reasons for deciding on this API, the block model has an extra thread for the read/write requests to be executed in its own context. The public API for read/write operations works by submitting block requests to the block device queue which are pulled and executed in the block context. We want to delay the execution of the IO requests in the block context as the virtual machine loop—where the cpu instructions run—is single thread. If the IO operations were executed in the same context as the virtual machine loop, then the whole system would get stuck.

LBA 28-bit Addressing

Logical Block Addressing defines the addressing of data on the disk device by the linear mapping of sectors. This means the blocks are located by an integer index, with the first block being LBA 0, the second LBA 1, and so on. The read/write commands, whether the data transfer protocol is either PIO or DMA, use 28 bits for addressing one sector. Hence the maximum size supported by the 28-bit addressing is 228 × 512 bytes = 128 GB, since the size of one logical sector is 512 bytes. In order to support LBA 28-bit addressing, the ATA standard uses the LBA group registers and the first 4 bits in the Device register with the following mapping: LBA Low=LBA(7:0), LBA Mid=LBA(15:8), LBA High=LBA(23:16), Device(3:0)=LBA(27:24).

Implementation

This section begins with the initialization phase of the ATA emulator and continues by presenting some software protocols like reset, PIO in/out, DMA, and how the master/slave device addressing is handled in the implementation. At the end we present the ATA commands that have been implemented till now.

Initialization

The initialization of the ATA adapter begins in the pci_ata_init function by opening the backing file as disk storage. The PIO and DMA commands that will be handled by the emulator will result in read and write calls to the backing image file descriptor. Each IDE controller appears as a device on the PCI bus. If the class code is 0x01 (Mass Storage Controller) and the subclass code is 0x1 (IDE), this device is an IDE Device. The PCI Adapter implements a subset of the PCI standard type configuration header register set. First, the PCIR_DEVICE and PCIR_VENDOR registers are set with 0x8211 and 0x1283 values—a Waldo ATA Controller. The PCI Class code must be configured by setting the Base-class, Sub-class Codes with the next values: 01h – Mass Storage and 01h – IDE. The Programming Interface Code should be correctly handled by setting the PCIP_STORAGE_IDE_MASTERDEV bit to indicate that the adapter can do bus master operation, and either the PCIP_STORAGE_IDE_MODEPRIM or PCIP_STORAGE_IDE_MODESEC bit to select the ATA channel. In the initialization function of

the ATA Host emulator, the BAR registers must be allocated. The IDE device only uses five BARs of the six. The PCI BAR0 and BAR2 represent the base addresses of primary and secondary channels. The PCI BAR1 and BAR3 represent the base addresses for the control register for primary and secondary channels. The PCI Base Address Register BAR4 is the base address of the ATA Bus Master I/O registers. The initialization is completed by reserving the proper IRQ numbers for each ATA channel.

Device Addressing Considerations

When a value is written to the registers of both devices, the host discriminates between the two by using the DEV bit in the device register. Data is transferred in parallel either to or from host memory to the device's buffer under the direction of commands previously transferred from the host. The device performs all operations necessary to properly write data to or read data from the media. When two devices are connected on the cable, commands are written in parallel to both devices, and for all except the EXECUTE DEVICE DIAGNOSTIC command, only the selected device executes the command. When the device control register is written, both devices respond to the write regardless of which device is selected. When the DEV bit is cleared to zero, Device 0 is selected. When the DEV bit is set to one, Device 1 is selected. When two devices are connected to the cable, one is set as Device 0 and the other as Device 1 [6].

Software Reset Protocol

At some point the guest driver is going to software reset the ATA controller. One point would be when the guest driver is looking for any signs of ATA/ATAPI present in the channel. To reset the controller, the driver writes the ATA_A_RESET bit in the ATA_CONTROL register. For emulation of this operation, the ATA emulator sets the ATA_E_ILI bit in the ATA_ERROR register and the ATA_S_READY bit in the ATA_STATUS register. For more details and a better understanding please take a look at the ata generic reset function in the `sys/dev/ata/ata-lowlevel.c` FreeBSD driver.

PIO Data-in Command Protocol

The PIO data-in protocol is used to transfer one or more blocks of data from the disk device to the host memory. It is closely related to the commands that use this protocol since they are responsible for preparing the PIO data buffer. The use of this protocol is transparent for the ATA driver since it needs to specify only the parameters for a specific PIO command. Hence there is a PIO data-in class containing commands that use this protocol. Such commands are: IDENTIFY DEVICE, READ MULTIPLE, READ SECTOR(S). After one of these PIO commands is issued by the ATA guest driver, the emulator reads the data from the block disk into the PIO buffer and interrupts the host, which means the data is ready to transfer. At that moment, the host starts to transfer data by polling the DATA register and getting 4 bytes at each read. When the buffer is empty, the PIO command is completed. The host gets interrupts for each transferred block which is 128 sectors by default. If the total count of sectors is not evenly divisible by the block count, the emulator interrupts after the last partial block has been transferred.

PIO Data-out Command Protocol

The PIO data-out protocol is used to transfer one or more blocks of data from the host memory to the disk device. Differing from the PIO data-in protocol where the data buffer is prepared by the emulator before the transfer starts, the host writes the data into the buffer. It is closely related to the commands that use this protocol as they are responsible for transfer parameters like the sector count and the offset in the block disk. There is a PIO data-out class containing commands like WRITE MULTIPLE, WRITE SECTOR(S) that uses this protocol. After one of these commands is issued, the emulator saves the transfer parameters (sector count and offset) and waits for the data from the host without asserting the interrupt. The ATA driver starts to transfer data by polling from the DATA register into the PIO buffer adding 4 bytes at each write. When the total number of sectors has been written on the block disk, the PIO command is completed. For every transferred block, the emulator writes the data on the block disk and interrupts the host. If the total count of sectors is not evenly divisible by the block count, the emulator interrupts after the last partial block has been transferred.

DMA Command Protocol

The DMA protocol is used to transfer one or more blocks of data either from the host memory to the disk device, or from the disk device to the host memory. It is closely related to the commands that use this protocol since they are responsible for the transfer parameters like the sector count and the offset in the block disk. There is a DMA class containing commands like READ DMA, WRITE DMA that uses this protocol. Before further explanation, we want to introduce the concept of DMA transaction. Basically the DMA transaction contains three parts: the READ/WRITE DMA command, the start, and eventually the stop of the DMA procedure. After one of these commands is issued, the host driver starts the DMA transaction by giving the address of the Physical Region Descriptor Table to the emulator and setting the Start/Stop Bus Master bit in the ATA Bus Master Command Register.

The emulator gets the physical address in the guest memory space and needs to translate it into the bhyve process memory space. To achieve this, the emulator uses the paddr_guest2host function which uses the guest address as a parameter and returns a pointer to the host memory space. The PRD Table contains several Physical Region Descriptors (PRDs) which describe areas of the guest memory that are involved in the data transfer. Each PRD entry has 8 bytes and specifies the location where the data will be transferred to/from. The first 4 bytes represent the address of a physical guest memory region; the next 2 bytes specify the number of bytes that are supposed to be transferred to/from that region. The physical address of the entry is translated into the bhyve address space in the same way as the PRDT address using the paddr_guest2host. If bit 7 of the last byte is set, that is the end of the table. At this moment the emulator iterates through the entries to prepare the block request that will be executed by the block instance. Basically a block request contains an array of iovec structures with each iovec structure indicating an entry in the PRD Table. After the block request has been prepared, it is executed in the context of the block instance, meaning a write or read operation on the file descriptor associated with the block. The guest driver is notified when the transfer is complete by asserting an interrupt as it has to initiate the last phase of the transaction by clearing the Start/Stop Bus Master bit in the ATA Bus Master Command Register. At that moment, the emulator verifies the state of the transaction and sets the status register indicating whether the transfer has completed successfully or not and eventually marks the transaction as completed.

Command Descriptions

In FreeBSD, ATA commands are implemented in the `sys/dev/ata/ata-lowlevel.c` driver. This driver is directly controlling our ATA controller emulator. For a better understanding of the command structure, we took a look at some functions that are related to ATA commands in the guest driver: ata_generic_command, ata_tf_write, ata_wait.

An ATA command starts by selecting the master/slave drive, waiting for the controller to clear the ATA_S_BUSY register. When it is ready to issue the command, the driver enables the interrupt by setting ATA_A_4BIT bit in the ATA_CONTROL register and continues with some write register operations such ATA_FEATURE, ATA_COUNT, ATA_SECTOR, ATA_CYL_LSB, ATA _CYL_MSB, ATA_DRIVE to configure the parameters of the command. Actually the ATA command is issued to the controller by writing code into the ATA COMMAND register. The emulator saves these parameters into its internal data structures and uses it to find the meaning of the command. After the command is fully emulated, an interrupt should be asserted to notify the guest driver.

The commands implemented in the ATA emulator meet the general feature set commands supported by the ATA 6 standard. We describe some of the implemented ATA commands below.

• **IDENTIFY DEVICE:** This command enables the host to receive parameter information from the device. The device sets the BSY bit to 1, prepares to transfer the 256 words of device identification data to the host, sets the DRQ and READY bits to 1, clears the BSY bit to zero, and asserts INTRQ. The host may then transfer the data by reading the data register using the PIO data-in protocol. The data is transferred using 128 successive word transfers. The order of bytes inside the word is different between the driver and emulator memories. For example,

to send the *FaK3 MoDeL IDA diSk* string to the host memory, the emulator prepares the *aF3KM DoLeI ADd Si k* string. Using this command, the host finds out the CHS parameters of the disk device like the number of cylinders, heads, and sectors, the model name, serial number, and firmware version, and also the capabilities supported. The capabilities supported by the ATA emulator are: both Multi Word DMA2 and the PIO4 data transfer protocol working with 28-bit LBA addressing, support for write-read verify, and flushcache.

• READ MULTIPLE: This command is issued by the ATA driver to read data using the PIO data-in protocol. It specifies the number of sectors to be transferred and the offset on the block disk using the LBA 28-bit addressing mode. The emulator reads data from the disk, prepares the PIO buffer, marks the PIO read command in progress and interrupts the host meaning the data is ready. After the host gets the interrupt, it starts to transfer the data.

• WRITE MULTIPLE: This command is issued by the ATA driver to write data using the PIO data-out protocol. It specifies the number of sectors to be transferred and the offset on the block disk using the LBA 28-bit addressing mode. The emulator saves the command parameters into the PIO setup, marks the PIO write command in progress, and waits for the host to transfer the data.

• READ AND WRITE DMA: These commands are issued by the ATA driver to read / write using the DMA data protocol and they indicate the first phase of the DMA protocol. The protocol specifies the number of sectors to be transferred and the offset on the block disk using the LBA 28-bit addressing mode. The emulator saves the command parameters into the DMA setup including the direction of the operation (read / write), and marks the DMA transaction as started. Afterwards the host prepares the PRD Table and activates the DMA transaction.

• FLUSH CACHE: This command is a non-data ATA command used by the host to ask the device to flush the write cache. Basically the emulator creates a flush request to the block device which flushes the file descriptor associated with the disk drive.

LISTING 1. ATA LEGACY DRIVER'S LOG

```
atapci0: (ITE IT8211F UDMA133 controller)
port 0x2020-0x2027,0x2028-0x202b,0x170-0x177,
0x376,0x2040-0x204f at device 3.0 on pci0
ata0: <ATA channel> at channel 0 on atapci0
ata1: <ATA channel> at channel 1 on atapci0
ada0 at ata0 bus 0 scbus0 target 0 lun 0
ada0: <BHYVE ATA IDE DISK 1.0> ATA-6 device
ada0: Serial Number 123456
ada0: 16.700MB/s transfers (WDMA2, PIO 65536bytes)
ada0: 8192MB
ada0: (16777216 512 byte sectors: 16H 63S/T 16644C)
ada0: Previously was known as ad0
ada1: at ata0 bus 0 scbus0 target 1 lun 0
ada1: <BHYVE ATA IDE DISK 1.0> ATA-6 device
ada1: Serial Number 123456
ada1: 16.700MB/s transfers (WDMA2, PIO 65536bytes)
ada1: 8192MB
ada1: (16777216 512 byte sectors: 16H 63S/T 16644C)
ada1: Previously was known as ad1
```

Scenarios and Results

Using *-s 3:0,ata-hd,0,./diskdev_ata,./diskdev_ata_slave* input for the bhyve application, the next output is printed by the guest ATA legacy driver (see Listing 1).

We observe that the ATA controller is successfully recognized by the atapci driver and the BAR addresses are properly allocated (BAR1 = 0x2020-0x2027, BAR2 = 0x2028-0x202b, BAR3 = 0x170-0x177, BAR4 = 0x376, BAR5 = 0x2040-0x204f). Also both ATA channels are handled by the ata0 and ata1 drivers. Because both disk drives have been specified in the input string, indicating the first one as master drive and the second one as slave drive, there are two instances of the ada driver—ada0 and ada1. Hence, at the Partitioning phase of the FreeBSD Installer there are two disks on which to install the FreeBSD virtual machine with both of them being supported (see Listing 2). The model implemented by the ATA emulator is an ATA-6 device which supports both PIO and WDMA2 data transfer protocols. It should be emphasized that the speed of 16.700MB/s is not the actual speed of the data transfer between the guest operating system and emulator, but it represents the speed required by the WDMA2 standard.

LISTING 2. FreeBSD INSTALLER—PARTITIONING

```
Select the disk on which to install FreeBSD.
vtbd0     654 MB Disk
ada0      8.0 GB ATA Hard Disk (BHYVE ATA IDE DISK)
ada18.0 GB ATA Hard Disk (BHYVE ATA IDE DISK)
```

LISTING 3. ATA TESTING

```
dd bs=$BLOCK_SIZE count=1
if=/dev/random of=tests/testX
dd bs=$BLOCK_SIZE count=1
if=tests/testX of=/dev/ada1 oseek=$MAX_SECTORS
reboot
dd bs=$BLOCK_SIZE count=1
if=/dev/ada1 of=out/testX iseek=$MAX_SECTORS
diff out/testX tests/testX
```

At the moment, the guest FreeBSD virtual machine can be installed on either ada0 or ada1 and played without any restrictions using the ATA emulator. The only limitation is the size of the disk, which is maximum 128 GB due to LBA 28-bit addressing.

Even though the full installation of the virtual machine on the ATA disk proves the correctness of the ATA emulator, a set of tests for a proper validation have been performed. Using a similar approach to Listing 3, the BLOCK_SIZE and MAX_SECTORS variables have been varied to cover the whole disk with different chunk sizes. All the tests were passed proving the correctness of the implementation.

As we said before, the maximum transfer rate specified by the ATA-6 standard using the DMA Multi-word 2 is 16.7MB/s. However, this value seems smaller than the actual speed of our emulator since it is for hardware devices. In order to get the transfer rates of the ATA emulator, we use the diskinfo tool running inside the virtual machine which has the device emulated. Using the 'diskinfo -t /dev/ada1' command, where /dev/ada1 is the ATA disk emulated by our implementation, we get the following results seen in Listing 4, indicating transfer rates of more than 100MB/s, which is more than enough. The ATA-6 has progressed with Ultra DMA giving data transfer rates up to 100MB/s, which is comparable with our emulator using WDMA2. Hence our implementation is compliant with ATA-6 requirements without a need to develop the Ultra DMA feature.

Even though the ACHI and ATA disk drive controllers are completely different from each other and do not use the same transfer data protocols, it would be interesting to measure the performance of the AHCI module and compare it with our ATA emulator. When we do the

same procedure using the diskinfo tool on a partition controlled by the AHCI driver, we get the following transfer rates displayed in Listing 5.

These results are expected since the AHCI standard uses the UDMA6 data transfer protocol which imposes transfer rates of 300MB/s for hardware devices with more than the 16.7MB/s transfers specified by the WDMA2 data protocol. But the main reason why the AHCI emulation has a better transfer rate is that it uses the LBA 48-bit addressing mode, which makes it possible to transfer more data sectors (65536) per DMA transaction with less overhead in the communication between the host driver and the disk emulator.

LISTING 4. TRANSFER RATES

```
outside: 102400 kbytes in 0.719681 s = 142285 kB/s
middle:  102400 kbytes in 0.796385 s = 128581 kB/s
inside:  102400 kbytes in 0.781721 s = 130993 kB/s
```

LISTING 5. AHCI TRANSFER RATES

```
outside: 102400 kbytes in 0.326701 s = 313436 kB/s
middle:  102400 kbytes in 0.339824 s = 301332 kB/s
inside:  102400 kbytes in 0.348496 s = 293834 kB/s
```

Conclusion and Further Work

In this article we presented a model of emulation for an ATA disk drive controller under the PCI attachment. We managed to integrate it with bhyve and implement the general feature set commands supported by the ATA-6 standard. The implementation was tested for validation and the performance was measured and compared against the AHCI emulation. There is another option as the ATA emulator could be a child of the PCI-ISA bus. The utility of ATA would be hugely improved in legacy operating systems given that the boot loaders would be able to use it since the IO ports and IRQs would be at fixed locations. Future work aims to achieve ATA emulation as a PCI-ISA attachment and merging it to the final code base. ●

ALEX TEACA graduated from the University Politehnica of Bucharest and is currently a second-year master's degree student focused on parallel and distributed computer systems. His departmental research project involved work on the ATA/ATAPI emulation in bhyve with Mihai Carabas.

MIHAI CARABAS is a PhD student at the University Politehnica of Bucharest studying virtualization. He has contributed to the FreeBSD Project and DragonFlyBSD virtualization code.

PETER GREHAN is a FreeBSD committer who has been using BSD-derived operating systems in some form since the days of DEC Ultrix. He co-developed and maintains the bhyve hypervisor with Neel Natu.

References

[1] J. Boyd. Serial ata advanced host controller interface, page 9. Intel Corporation, Hillsboro, Oregon. (2008)

[2] T. Goodfellow. Ata/atapi host adapters standard, page 15. Pacific Digital Corporation, Irvine, California. (2003)

[3] T. Goodfellow. Ata/atapi host adapters standard, page 11. Pacific Digital Corporation, Irvine, California. (2003)

[4] P. T. McLean. Information technology–at attachment with packet interface–6, page 16. American National Standards Institute, New York, New York. (2002)

[5] P. T. McLean. Information technology–at attachment with packet interface–6, page 63. American National Standards Institute, New York, New York. (2002)

[6] P. T. McLean. Information technology–at attachment with packet interface–6, pages 56–57. American National Standards Institute, New York, New York. (2002)

[7] Wikipedia. Parallel ata–wikipedia. http://en.wikipedia.org/wiki/Parallel ATA, 2015. (Online; accessed February 14, 2015)

[8] Wikipedia. Serial ata–wikipedia. http://en.wikipedia.org/wiki/Serial ATA, 2015. (Online; accessed February 14, 2015)

SEE TEXT ONLY

conference REPORT™

by Warren Block

AsiaBSDCon (Amazingly Friendly People)

The 2016 AsiaBSDCon conference was held in Tokyo from March 10 through March 14, 2016. If I had expected my paper on the new FreeBSD documentation translation system to be accepted, I would have developed even a small Japanese vocabulary. But it was a pleasant surprise.

WARREN BLOCK PHOTO

THE TRIP

Getting to Japan required a bit of travel. The good news was that the Boeing 787 has higher air pressure and humidity than other planes, so it is more comfortable. The bad news was that the flight would take 11-1/2 hours. Tip for travelers: take your own over-the-ear headphones, and watch some movies to pass the time. Due to the long flight and the extreme difference in time zones—somewhere between 8 and 14 hours, I'm still not sure—the times and events given here even if numerically inaccurate will still capture the feeling of the event.

I had been outside the U.S. before, to BSDCan in Ottawa. Still, Canada is hardly the same as Japan. Concerns about difficulty with customs turned out to be entirely unfounded. There was nothing more difficult than standing in line, and eventually I found the train station at the Narita airport.

The Narita Express is a fast train from the airport west into Tokyo itself. It is a train trip that would convince even the most cynical American of the value of train travel: big comfortable seats; a smooth, quiet ride; video and audio announcements of location and destination in Japanese and English. The video monitors also showed news clips and continuous ads for some remarkably ugly sports shoes.

The Narita Express only goes to the Tokyo subway station, so it was necessary to transfer to the local subway trains to reach the hotel. The Tokyo station is huge, at least three different levels, and I arrived there during rush hour. After several circumnavigations of the entire station, several locals took pity on me and the correct train was located.

This was the only difficulty I had, because, while almost all of the station signs are bilingual, the particular ones I needed were Japanese only. Many Japanese people understand individual English words even if they do not speak English. And many are completely fluent.

Eventually, I arrived at the Ichigaya station near the hotel. A handheld GPS was useful several times, because it was both dark and raining.

The whole process would be easier for someone more used to passenger trains or subways. In the American West, trains carry coal, grain, cows, and occasionally the powdery dry version of slimy bentonite, but almost never people. The Tokyo system is very smooth. Users pre-load an RFID "Suica" card with cash, then scan it at entrances and exits. When lost, merely look like a stupid tourist (probably my default look) and people will help you.

Check-in at the hotel was very easy; the AsiaBSDCon organizers had it all set up. However, the room had two-prong outlets, and my notebook AC adapter had a three-prong plug. Japan is apparently in the middle of the two-to-three changeover. A U.S.-style cheater plug, ground cut off as per tradition, will create ungrounded fun for all. The Japanese adapter I bought actually had an insulating boot over the wire.

An additional difference was that many of the available wireless networks were still using WEP. This was not obvious from the 'ifconfig wlan0 list scan' output, and only became clear after someone mentioned it at the conference. The hotel room did have wired Ethernet, though, with its welcome simplicity.

THURSDAY

Many FreeBSD conferences are preceded by developer summits, meetings for planning and coordinating to take advantage of the rare facetime with other developers. There is a small but dedicated group of documentation people, and that group was mostly present again at this conference. Dru Lavigne, Benedict Reuschling, Brad Davis, and, of course, AsiaBSDCon organizer Hiroki Sato were present. Allan Jude was there too, but he has his hands in so many things that we rarely see him.

There is always a larger contingent of source and ports developers, and frequently some OpenBSD developers. If we are lucky, NetBSD and DragonFly people are present as well, and they were at AsiaBSDCon. This mix of projects with different goals adds variety, and often results in cross-pollination where projects share their solutions or techniques.

The FreeBSD developer summit's biggest topic of discussion was "pkgbase," packaging the base system like a collection of ports. This work is proceeding, and many people are curious about implementation details. Documenting how it will work is also a big concern, as are other new features of FreeBSD 11. We also talked about help with documentation, particularly review. The sad fact is that once a person learns about something, they usually don't go back to the Handbook later to review those sections for accuracy and completeness. The flip side is that we documenters have not been very good about announcing major revisions and additions, so people are not always aware when things have changed. We have some ideas in progress on this. We also repeated that source developers only need to worry about content in their attempts at documentation. If they want to use DocBook or mdoc markup, that is great. But if they just want to work with text, documentation project people can help with the markup.

Bento box lunches were provided by the conference, with an interesting variety of dishes. The organizers had provided vegetarian dishes, which was very nice of them and probably took some effort. Fish is used in a great number of foods in Japan. Some of my rice had tiny, transparent fish staring back at me, but that was either a mistake or quite possibly a garnish.

Later that day, several projects gave status reports. They were all interesting, but Alistair Crooks's reports on NetBSD and pkgsrc were particularly comprehensive and well-presented. It is always nice to hear these kind of reports from people who are both experts on their subjects and good speakers.

FRIDAY

The doc developers had our own meeting later where we talked about several ongoing subjects like the new PO translation system, the perennial website bikeshed, and our lack of communication about all the things that are happening with the documentation. The ports team has done a very good job communicating all the interesting things they are doing, and we need to do the same thing.

In the afternoon, we made a brief excursion to Akihabara, the fabled district of Tokyo where electronics from the smallest component parts to the largest assembled items are sold. There is a staggering amount of electronics crammed into every available area. Some places are one building with a bunch of tiny, specialized booths sharing the space. I had hoped to find a particular Japanese-made crimping tool, but it seemed unlikely. In the first store we visited, one of the booths sold only crimping tools, and the one I wanted was hanging on the wall! Information overload is easy to experience in Akihabara, and we returned to the conference for a later session.

The documentation tutorial began in the late afternoon. I attended that, mostly to see how well the new translation system worked. Dru and Benedict were the teachers. Over the three-hour session, they worked with Daichi GOTO to cover all the aspects of working with FreeBSD documentation. A few rough spots in our examples were discovered, but at the end, GOTO-san had completed a Japanese translation of the leap seconds article.

We did discover that a few seats in that room were somehow the focus of vibrations from heating or air-conditioning equipment. Sitting in those seats was much like riding in a car with a couple of wildly out-of-balance wheels. This was useful knowledge for later talks that were held in the same room.

SATURDAY

The first round of conference presentations began on Saturday.

Kamil Czekirda gave his presentation on "FreeBSD Test Cluster Automation." This was a separate project from the existing test cluster, and used network booting to completely set up raw machines as test nodes. It was an intriguing setup, and tied in well with automated testing we had discussed earlier at the dev summit.

"High Density Filers" was Baptiste Daroussin's talk on setting up large fileservers that are booted from the network. They had tried several different operating systems, and FreeBSD had done very well in the comparison, shown with benchmarks and pretty rigorous testing. Remaining issues were ones that many of us have experienced with the FreeBSD PXE boot loader and other network-booting problems. Improvements to these components will make FreeBSD more versatile and competitive with standard Linux boot loaders like Syslinux and Grub.

During a lunch break, I walked to a couple of local bike shops looking for water bottles with their shop logos. This turns out to be something that the entire world stopped doing, so neither had such a thing. On the way back, I saw a shop called "Leonidas Chocolates" and thought it would be nice to take something back for others at the conference. Three women were working there, and after I asked for an assortment, they wanted to know if it was for a gift. "Sure," I said, thinking that was probably all my vocabulary would allow. All of their faces lit up. Six pieces of chocolate were carefully wrapped and presented for my inspection. The box was carefully taped, and then I had a choice of ribbon color. Thinking it was the safest choice, I asked them to choose. All of them became even more interested, and they chose a red ribbon. The box was carefully placed in a fancy bag. It's still not really clear, but I might be engaged to one or even all three of those women.

A banquet was held Saturday night, conveniently located at my hotel. This was a very upscale event, with lots of food, a quiet enough atmosphere to speak to others, and an open bar that encouraged speaking to others. I sampled "shochu," a distilled liquor made from barley. Sake is direct and obvious about its intentions. This shochu was subtle to the point of misdirection.

SUNDAY

At last, it was time to give my own presentation, a fairly innocuous little number about the new PO translation system for documentation, why translators are important to FreeBSD, and how this new system can help them. More people attended than I had expected. While there might have been some shochu-induced overhead, most people looked fairly alert. The projector did not even put up a fight. I forgot only 20% or 30% of the points I had intended to make. The audience all appeared to be awake at the end, a rousing success by some standards (i.e., mine). The topic filled the time available, avoiding the embarrassing silence of finishing very early or the awkward arrival of people for the next presentation.

After my presentation, I could relax and enjoy the others. Allan Jude talked about his work on booting FreeBSD from encrypted disks and the patently ridiculous multi-stage loading that FreeBSD has used for ages. It was an entertaining and insightful talk.

The conference keynote was given by Stephen Bourne. He spoke entertainingly at BSDCan last year about how he wrote the sh shell. At AsiaBSDCon, it was about the development of Unix, how and why things came to be the way they are. He is an excellent speaker, and it is well worth attending his presentations.

Finally, I went to Kirk McKusick's "Brief History of the BSD Fast Filesystem." It struck a pleasant balance between technical details and overview, and gave me new respect for UFS, which not only takes far less resources than ZFS, but is a very good filesystem at the same time.

There was a Work in Progress session, where people gave very quick reports on ongoing projects, then the traditional FreeBSD Foundation drawing. Three prizes had been donated for the Foundation to give away. Two were AMD-based router systems with multiple Ethernet ports, one a bare board and the other in a steel case that would safely support a car. The third was a Minnowboard Turbot donated by Netgate. This is a 64-bit Atom processor with 2G of RAM, USB 2.0 and 3.0 ports, SATA, HDMI, and gigabit Ethernet, all in a 3 x 4-inch single board computer. Also included was an aluminum case with a special laser-cut AsiaBSDCon commemorative graphic. I never win anything—but I won the Minnowboard!

THE RETURN

The return trip was somewhat easier. Brad Davis managed to upgrade my ticket to an "economy plus" seat with more legroom, and in an otherwise-empty row. At the airport, we waited in the United lounge, where throngs of servants peeled grapes for us, disposed of empty edamame pods, and then eventually escorted us to the plane in an elephant-mounted howdah.

The return flight took only nine hours, still appreciable but tolerable, and much more comfortable, thanks to Brad.

There was a short layover at the Denver airport. Everyone who has waited in an airport knows the value of an electrical outlet. Some fancier seats have them nearby or even built into the seat frame. But gate B56 had not one single outlet. A vigorous search and I finally spotted one over in the corner on some metal trim. I quickly relocated to the seat next to it, unpacked my AC adapter, cables, and computer. Balancing all this, I turned to the outlet to discover that it was not an outlet, but instead a photo-realistic sticker of an outlet, placed there by some amazingly evil bastard. People wondered what I was laughing at and edged away nervously.

RECOMMENDATIONS

- Go to AsiaBSDCon.
- Take a two-prong AC adapter or cheater plug.
- The first time, meet up with someone who has been there before to help navigate.
- Have waypoints set in your GPS or phone for the conference location, hotel, and train station.
- A small vocabulary of Japanese words can go a long way. "Sumimassen" (excuse me) and "domo arigato" (thank you very much) cover many situations.
- Take a small umbrella. It will rain.

SUMMARY

The conference had many more presentations and meetings than I've described here. The entire event was an adventure before, during, and after the conference. Go to AsiaBSDCon. It is awesome. ●

WARREN BLOCK has been using FreeBSD since 1998, and has been a documentation committer since 2011.

SEE TEXT ONLY

this month In FreeBSD

BY DRU LAVIGNE

The Essen Hackathon

This month, we speak with Benedict Reuschling, who recently participated in the second Hackathon in Essen and who also spoke at the Open Source Datacenter Conference in Berlin. In addition to teaching a UNIX for software developers class at the University of Applied Sciences, Darmstadt, Germany, Benedict stays busy as a FreeBSD documentation mentor, a member of the FreeBSD German translation team, a Director at the FreeBSD Foundation, and a proctor for the BSD Certification Group. He provides some interesting insights about the importance of face-to-face hackathons and for participation in non-BSD conferences.

The second FreeBSD hackathon was held April 22–24 at the Linuxhotel in Essen, Germany. We had people from Canada, Sweden, France, Belgium, and the Netherlands, with some recurring participants.

I picked up Allan Jude from Frankfurt Airport and together we took the train to Essen. At the train station, we were picked up by someone who took part in the BSDA certification exam later that afternoon and we drove to the Linuxhotel. We dropped off our luggage and while I proctored the BSDA exam, Allan started hacking on his implementation of the Skein checksum algorithm (https://reviews.csiden.org/r/223/) for FreeBSD's ZFS that he wanted to make available. Lars Engels, the main organizer, arrived soon after and managed a couple of things with the Linuxhotel staff. After the BSDA exam was over, Lars and I went to buy some stuff for the barbecue later that evening. We were joined by Christian Brueffer and on the way there, we had a good exchange about what's new since the last time we saw each other.

After we got back and the afternoon progressed, more people arrived who had not attended before. We showed them around and started preparing the barbecue. Some other participants were stuck in traffic and joined us a while later when the barbecue was already going on, but everyone did get something in the end. This was a good chance to get to know each other, and people were soon talking about the latest developments in FreeBSD, as often happens when the right people come together. When it was getting dark, we all met in the room assigned to us by the Linuxhotel staff for beers and drinks to talk some more. Around midnight, tired from their travels, most people went to sleep.

On Saturday, which was the main hacking day, we met for breakfast (there was another group there from the Serendipity project) and then met in the hacking room to do some work. The room had everything we needed: cables, WiFi, chairs, tables, a projector, as well as a coffee machine and

fridge with various drinks. Lars and I welcomed everyone a second time, talked about some organizational things, and mentioned our sponsors. The owner of netzkommune.de (who also sponsors BSDCan and will go there for the first time this year) sponsored everyone's accommodation and joined us in the afternoon. As representative of the FreeBSD Foundation, I said a few words about our efforts in Europe and we then distributed the swag. The backpacks and bags were received with applause by the surprised participants.

After that, we had a short impromptu presentation about how the shell works and processes inputs by Jilles Tjoelker. We then continued to talk about some of the things we had as agenda points. After that, people hacked on their own little projects they had brought with them or formed small groups to work together. Commits were marked with a "Sponsored by: Essen Hackathon 2016" (http://freshbsd.org/search?q=+hackathon+2016+%21openbsd+%21bitrig+%21pkgsrc+%21netbsd+%21edgebsd+%21pfsense+%21pcbsd+%21hardenedbsd).

For lunch, we ordered pizza and hacking continued well into the late afternoon. Sometimes we were interrupted by bigger discussions relevant for everyone; sometimes we only communicated via IRC. After our netzkommune.de sponsor arrived and we thanked him for his generosity, we carpooled to that evening's restaurant. The food was great and we shared many stories over dinner and drinks. When we came back, we all gathered in the breakfast room on comfy couches to talk some more, while some others continued hacking.

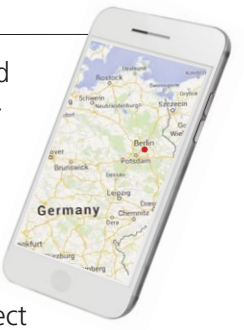
The next morning, I found out that some people stayed there until 1 a.m., while most people went to bed much sooner than that. After breakfast, we once again gathered in the hacking room to make some more progress before we had to leave in the evening. Doner kebab was ordered for lunch and a couple of hours after that, people started leaving one by one. Overall, people were very happy with the location, equipment, accommodation, food, and drinks, as well as the evening event. They encouraged us to do such an event again next year and said they will come back then for some more hacking.

Open Source Datacenter Conference

The Open Source Datacenter Conference (OSDC, https://www.netways.de/osdc/) took place April 26–28 in Berlin, Germany.

Overall, OSDC is a very well organized conference. We were a bit stunned by the entry price for regular participants of 899 Euro. This includes the hotel stay, but is still pricey if you are an open source person not associated with a company and want to inform yourself on things like DevOps, Jenkins, Puppet, Cloud Infrastructure Management, and everything else you might need in a Datacenter context. It is clearly intended for open source users in companies that can afford to send their employees there. The talks are recorded and the slides are available from previous years. It is not a marketing event, besides the "we're hiring" and booth area from a few of the sponsors.

Allan Jude and I gave a presentation on ZFS and it was the only one focusing on storage and FreeBSD. Our talk was well attended. We asked at the beginning how many people in the audience knew about ZFS, and a couple of hands went up. The talk went well, and we had a couple of good discussions afterwards. It is clear that the whole GPL/CDDL licensing issue and Ubuntu's course of action create some FUD in the Linux world and results in a "wait and see" attitude, while we advertised the availability and stability of ZFS on FreeBD.



In the afternoon, we had a good exchange with Colin Charles about MySQL. He was very glad about the state of MySQL on FreeBSD and mentioned the good relationship that the MySQL project has with the FreeBSD committer for these ports.

We spoke with one of the organizers who said that the high entrance fee for non-speakers has been bothering them as they also want more non-commercial attendees and open source enthusiasts. They said that they were glad that we were there to represent a non-Linux operating system. They encouraged us to submit again next year.

Next year's conference is May 16–18, 2017, and I would encourage BSD people to submit talks for it. Not simply to show presence and that there are alternatives available, but also because Berlin is a nice city to visit. Since everything is in English, you should have no problem talking to other attendees and getting around. ●

Dru Lavigne is a Director of the FreeBSD Foundation and Chair of the BSD Certification Group.

SEE TEXT ONLY

svnUPDATE

by Steven Kreuzer

Over the past few years we have started to see an increased interest in ARM, a family of reduced instruction set computing (RISC) processors. The explosion in popularity can be partly attributed to inexpensive, single-board computers such as the BeagleBone and Raspberry Pi, which are providing folks with a simple yet extremely powerful platform to experiment on. In production environments, IT professionals are also looking toward ARM to help scale their infrastructure while simultaneously attempting to reduce power and cooling costs.

We are starting to see ARM powering everything from phones and tablets to high-performance computing environments, and I expect its footprint to grow as the Internet enables our couches to tweet and refrigerators to friend your blender on Facebook.

While ARM is still not a Tier 1 platform, every day we are seeing massive amounts of work being done to improve ARM64 support from both hobbyist developers and companies with commercial interests in building ARM-based appliances with FreeBSD at the core. In addition to all that, a brand new IO scheduler has been committed along with several changes that increase ZFS performance and give you more control over how hard a user or process can hit your storage pools. If that wasn't enough support for Haswell, GPUs were recently introduced that now enable FreeBSD to provide improved graphics and better battery life on a wide range of laptop configurations.

New CAM I/O scheduler—

https://svnweb.freebsd.org/changeset/base/298002

This is a new scheduler that favors reads over writes and has the ability to throttle the write throughput to SSDs. This new scheduler also brings in NCQ TRIM support for those SSDs that support it. In addition to better catering to read-heavy workflows, this new scheduler keeps a lot more statistics than the default scheduler, which can be useful for knowing when a system is saturated and needs to shed load. At this time, it is not the default scheduler and there may still be some rough edges, but it has been in use at Netflix for over a year.

I/O Limits on ZFS—

https://svnweb.freebsd.org/changeset/base/297633

Four new resources—readbps, readiops, writebps, and writeiops—have been added to rctl that now allow you to set limits on disk i/o for a process, user, login class, or jail. This should be a welcome addition to any system administrator who has to share a machine with IOP-intensive applications or users.

Improved ZFS speculative prefetch of indirect blocks—

https://svnweb.freebsd.org/changeset/base/297832

Scalability of many operations on a wide ZFS pool can be limited by the requirement to prefetch indirect blocks first. Recently added asynchronous, indirect block read partially helped, but did not solve the problem completely. This change extends existing prefetcher functionality to explicitly work with indirect blocks. Before this change, prefetcher issued reads for up to 8MB of data in advance. With this change, it also issues indirect block reads for up to 64MB of data in advance, so that when it is time to actually read those data, it can be done immediately.

ARM64 copyinout improvements—

https://svnweb.freebsd.org/changeset/base/297209

Making use of wider load/stores when aligned buffers are being copied has introduced a massive performance boost on FreeBSD/arm64. Performing a simple test of using dd to copy 1G from `/dev/zero to /dev/null` shows an increase from 410MB/s to 3.6GB/s.

THE INTERNET NEEDS YOU

GET CERTIFIED AND GET IN THERE!

Go to the next level with  **BSD CERTIFICATION**

Getting the most out of BSD operating systems requires a serious level of knowledge and expertise

NEED AN EDGE?

BSD Certification can make all the difference.

Today's Internet is complex. Companies need individuals with proven skills to work on some of the most advanced systems on the Net. With BSD Certification

YOU'LL HAVE WHAT IT TAKES!

SHOW YOUR STUFF!

Your commitment and dedication to achieving the **BSD ASSOCIATE CERTIFICATION** can bring you to the attention of companies that need your skills.

BSDCERTIFICATION.ORG

Providing psychometrically valid, globally affordable exams in BSD Systems Administration

Implement GELI in gptboot and gptzfsboot— https://svnweb.freebsd.org/changeset/bas(heading) e/296963

You may have noticed that the boot loader got a little fatter recently. Support for booting off GELI encrypted UFS and ZFS partitions was recently added. Additional details on how this was implemented can be found in a paper Allen Jude presented at AsiaBSDCon 2016.

Support for boot-time DTrace— https://svnweb.freebsd.org/changeset/base/297773

It is now possible to enable DTrace probes relatively early during boot before `dtrace(1)` can be invoked on `i386` and `amd64`. The desired enabling is created using *dtrace -A*, which writes a `/boot/dtrace.dof` file and uses `nextboot(8)` to ensure that DTrace kernel modules are loaded and that the DOF file describing the enabling is loaded by `loader(8)` during the subsequent boot. The trace output can then be fetched with dtrace -a.

Support tracing of userspace applications on ARM64— https://svnweb.freebsd.org/changeset/base/297611

The `dtrace_getupcstack`, which allows the function call stack to be captured, has been ported over to ARM64. This allows you to use DTrace to probe userspace applications.

Updated i915 GPU Driver— https://svnweb.freebsd.org/changeset/base/296548

Recently the FreeBSD graphics team undertook a massive initiative to update the i915 GPU driver to match Linux 3.8.13. The most exciting feature of this update is that it now brings support for Haswell GPUs to FreeBSD.

ARM64 kernel address space increased to 512GB— https://svnweb.freebsd.org/changeset/base/297914

This change, which also increases the DMAP region, has also been increased to 2TiB.

Support for 4 level pagetables on ARM64—https://svnweb.freebsd.org/changeset/base/297446

The userland address space has been increased to 256TiB.

Add kern.features flags for linux and linux64 modules— https://svnweb.freebsd.org/changeset/base/297597

To help third-party applications determine if the host supports linux emulation, a new kern.features flag has been introduced for linux and linux64 modules. If kern.features.linux is equal to 1, then 32-bit Linux binaries are supported, and if kern.features.linux64 is set to 1, 64-bit binaries are supported.

Updates to head/contrib

The base FreeBSD userland is made up of quite a few utilities, some of which are developed outside of the project. In the past few months we've seen a few updates to the third-party software that help create a great user experience.

- bmake has been upgraded to version 20160307— https://svnweb.freebsd.org/changeset/base/297357
- byacc has been upgraded to version 20160324— https://svnweb.freebsd.org/changeset/base/297276
- libxo has been upgraded to version 0.6.1— https://svnweb.freebsd.org/changeset/base/298083

STEVEN KREUZER is a FreeBSD Developer and Unix Systems Administrator with an interest in retrocomputing and air-cooled Volkswagens. He lives in Queens, New York, with his wife, daughter, and dog.

SEE TEXT ONLY

Building Community Around the Google Summer of Code

by Pedro Giffuni

FreeBSD has been a mentoring organization in each Google Summer of Code, GSoC for short, a program where Google sponsors college students to work on open-source projects. This is meant to be a great way to get students involved in software development and at the same time help fund development in communities that are usually short of resources.

In FreeBSD's case, we have been really fortunate. Not many projects get considered year after year as we have been. Part of our success comes, no doubt, from having a great project that is hugely influential and innovative. Another part of our success has come from having a consistent administrator behind it, namely Gavin Atkinson, and an amazing group of mentors who have been available to work with students.

I will admit that I became a FreeBSD committer after helping a student in his GSoC project. I was not the mentor; I just helped. The real mentor behind it thought I could be a good committer; and now, more than 1,300 commits later, and with my occasional screw-ups, I hope he still feels the same. The student didn't get a commit bit, but he did get a new job and eventually became a major contributor to a related open-source project, so I think the experience was pretty satisfactory for everyone involved. I have been a mentor for the Google Summer of Code for the last three years, and this year I became involved with GSoC administration.

So Is It Really Worth It or Are We Just Ripping Off Money from Google?

It is definitely worth it, but probably not in the way most people think. Code from students rarely makes it into FreeBSD's main code repository. FreeBSD has become a hugely complex project and achieving the code quality to just bring in new code is not easy. The Bugzilla database, if you are brave enough to look at it, has many open issues with patches that appear to fix them, but if we committed them blindly, we would open many more holes.

The GSoC brings students to work with a largely deployed and tested codebase and tasks tend to be challenging. As a FreeBSD developer, I have learned many things that are not taught in books, and in this sense, contributing to the project involves learning and growing as a professional.

Mentoring Projects

The key to any GSoC project is mentoring: we can't take on any project without mentors. As things go, we also don't have many trivial tasks for students to do, so having a mentor who thoroughly knows the existing codebase is essential.

I have been lucky enough to teach at the university level, and one thing that is similar in a GSoC project is that you learn to teach/mentor as you go along. As in real life, students don't always know what they say they know. As in real life, students and mentors don't always understand each other, and this is especially true if they don't see each other's faces. Projects can fail, but failures can be useful on their own.

My first project failed, and while I think that it was not really my fault, I still think we should not be afraid of failure. Both of us learned something in the process, and even when the student didn't agree he was not performing well, he later reapplied to another GSoC with us, so it was hardly a trauma he wouldn't recover from. On my side, I just keep learning new things every time I mentor.

The key behind a successful project is not really the code; the key is answering questions like: Is the student understanding and learning something? Is the student finding value from getting involved in the community? And the occasional, Wouldn't it be interesting if we were to try …?

Of course, the code might eventually come

about, and there is always a good chance someone else will find it useful. I have personally taken code from a GSoC developed in another project and imported it, after a lot of hard work, into FreeBSD. Eventually I reverted it and fixed it again and reimported it, but … well... if I had had to write the code from scratch, it wouldn't have happened at all.

We also try to introduce students into a culture: we have them use the same tools we use, we make heavy use of version control, code is reviewed, and interaction with other community members encouraged. It is disappointing to see how some other projects, which I won't detail here, finish a GSoC without a status report and no traces of the code, good or bad, that was developed as part of it.

Another thing we have noticed, and I find particularly interesting, is that when other projects, possibly but not necessarily a BSD variant, don't get selected for the GSoC, we may get an influx of developers with experience from those projects. This is great, as we introduce some important variety to our own development process: I would like it if we started working on common projects with other BSDs, but also with related codebase *Illumos*, or variants like *Debian kFreeBSD*, *UbuntuBSD*, or even different codebases like *Apache Cloudstack*. We surely haven't exploited all the possibilities.

How We Choose Them

Mentors are always the key: we have a steady influx of students each year, but developers don't just appear spontaneously. The selection process involves mentors voting for their favorite proposals, but ultimately we depend on finding mentors for the projects.

As we have made our way through the GSoC program, we have improved concepts like early mentorship, that is, getting potential students to discuss their proposals publicly and discussing them with possible mentors before the proposal is made. We do have a preference for students who have experience in the FreeBSD community and people who already have some experience in version control, be it subversion, git, or even perforce. It is also a good sign if the student has already used FreeBSD and knows how to rebuild the kernel or has already submitted a patch.

FreeBSD is, of course, rather particular in that it is a complete operating system: while it is not uncommon to find projects that affect only the kernel or only userland, it wouldn't be unexpected for a project to work on multiple areas to achieve its results. While we won't discard any general purpose project, we generally do try to favor projects that people can't do elsewhere, and so any work related to FreeBSD-specific technologies like *capsicum*, *netgraph*, *netmap*, *bhyve, jails*, or even somewhat less specific technologies like ZFS and DTrace are likely to catch our attention.

As of the time of this writing, we have received 36 proposals, and we accepted 15. We were conservative but I hope we caught the best of what our users want to find in FreeBSD.

Graduating from the GSoC

We have had students who participated multiple times and they have found it a great experience, which is ultimately what defines whether a project is fully successful or not. Recently Google has decided that students can only participate twice in the program. This is understandable and fair, and while we did lose a very interesting project due to it, we just have to remember that we are not here for the money; we are here because we like it. When there are exceptional projects, we will eventually find other ways to sponsor them.

Getting students involved in conferences and interacting with other developers, or even helping them get in touch with companies that use FreeBSD, is vital for the community and is a process that we are still learning.

HUGE thanks, Google!

Pedro Giffuni is a Mechanical Engineer from the Universidad Nacional de Colombia. He learned his way around computers when he was around 14 and trapped between the early days of Microsoft Basic in a Tandy Coco and an Intellivision. Pedro's first experience with FreeBSD was in 1997 when he installed 2.1.5-Release by downloading a floppy in his school in Bogotá from a then "fast" 19.2K modem connection. By then he knew how to program very well in Pascal and Fortran, which he would later discover to be basically useless. After much more fun than pain, Pedro has not needed to try other UNIX-like variants since then.

THROUGH JULY 2016 BY DRU LAVIGNE

2016 Events Calendar

The following BSD-related conferences will take place in June and July 2016. More information about these events, as well as local user group meetings, can be found at **www.bsdevents.org**.

BSDCan • June 8–11 • Ottawa, ON

https://www.bsdcan.org/2016/ • The 13th annual BSDCan will take place in Ottawa, Canada. This popular conference appeals to a wide range of people, from extreme novices to advanced developers of BSD operating systems. The conference includes a Developer Summit, Vendor Summit, Doc Sprints, tutorials, and presentations. The BSDA certification exam and the beta exam for the BSDP will also be available during this event. Registration is required.

SouthEast LinuxFest • June 10–12 • Charlotte, NC

SouthEast LinuxFest

http://www.southeastlinuxfest.org/ • The eighth annual SouthEast LinuxFest is a community event for anyone who wants to learn more about open source software. There will be several FreeBSD-related presentations and a FreeBSD booth in the expo area. There is a nominal registration fee for this conference.

Texas LinuxFest • July 8 & 9 • Austin, TX

http://2016texaslinuxfest.org • There will be a FreeBSD booth at Texas LinuxFest, to be held in the Austin Convention Center. The BSDNow crew will be in attendance and the BSD certification exam will also be available during this event. There is a nominal registration fee for this conference.